CITS 4009 Project 2 -Modelling and Clustering

ID: 23035606, Isaac Harris
ID: 22829076, Amal Seby Chirackal

Introduction

This dataset generated from kaggle website. https://www.kaggle.com/hadiyad/lendingclub-data-sets (https://www.kaggle.com/hadiyad/lendingclub-data-sets)

LendingClub is a firm that specialises in providing urban clients with various sorts of loans. When a firm receives a loan application, it must decide whether or not to approve the loan based on the applicant's profile. The bank's choice is connected with two categories of risks:
1) If the applicant is likely to repay the loan, the company will lose business if the loan is not approved.
2) Approving the loan may result in a financial loss for the firm if the applicant is unlikely to repay the loan, i.e. if he or she is likely to default.
The information provided includes information on previous loan applicants and whether or not they defaulted. The goal is to find patterns that suggest if a person is likely to fail on a loan, which may then be used to make decisions like rejecting the loan, lowering the loan amount, lending (to riskier applicants) at a higher interest rate, and so on. When a person receives a loan, there are two possible consequences.
1)Fully paid: The loan has been paid in full by the applicant (the principal and the interest rate)
2)charged-off: The applicant has failed on the loan since he or she has not paid the payments on time for a long period of time.

Lending to 'risky' candidates is the greatest cause of financial loss for the company, as it is for most other lending firms (called credit loss). When a borrower refuses to pay or runs away with money due to them, the lender suffers a credit loss.To put it another way, defaulting borrowers are the ones who cost lenders the most money.Customers who have been charged off are referred to as 'defaulters' in this situation. If these riskier loan applicants can be identified, the size of the loan can be lowered, reducing the amount of credit loss.The goal of this case study is to identify such candidates using machine learning.In other words, the firm needs to know the reasons (or driver variables) that cause loan default, i.e. the variables that are significant predictors of default.This knowledge may be used to the company's portfolio and risk assessment.

Loading libraries and Data

```
library(ggplot2)
library(readr)
library(lattice)
library(dplyr)
library(magrittr)
library(ggthemr)
library(skimr)
library(tidyverse)
library(rgdal)
library(openintro)
library(corrplot)
library(scales)
library(ggridges)
library(viridis)
library(hrbrthemes)
library(caret)
library(ebdbNet)
library(ROCR)
library(rapportools)
library(ggROC)
library(ROCit)
library(rpart)
library(rpart.plot)
library(ggpubr)
library(mlr)
library(wrapr)
library(pROC)
library(mltools)
library(ape)
library(data.table)
library(knitr)
```

```
# Load the data
lending_club_df<- read.csv(file = "lending_club_loan_two 3.csv", header = TRUE)
lending_club_df<- lending_club_df %>% mutate_if(is.character,as.factor)
lending_club_df$emp_title<-as.character(lending_club_df$emp_title)
lending_club_df$title<-as.character(lending_club_df$title)
lending_club_df$address<-as.character(lending_club_df$address)
lending_club_df$mort_acc[is.na(lending_club_df$mort_acc)] <- mean(lending_club_df$mor
t_acc, na.rm = TRUE)
lending_club_df <-lending_club_df %>% drop_na(revol_util, pub_rec_bankruptcies)
lending_club_df$state <- sub(".*\\b([A-Z]{2}) \\d{5}.*", "\\1", lending_club_df$addre
ss)
lending_club_df$state<- as.factor(lending_club_df$state)
```

Data Preparation

Part -1 Binary Classification

The response variable we have chosen is loan status.The loan status variable name is set to default. We assigned the labels 1 (charged-off) and 0 to these variable (Fully paid)

1)Fully paid(0): The loan has been paid in full by the applicant (the principal and the interest rate)
2)charged-off(1): The applicant has failed on the loan since he or she has not paid the payments on time for a long period of time.

Merging some Categorical Data that has so many levels

```
#Categorized income
lending_club_df$annual_inccat[lending_club_df$annual_inc < 50000] <- "Lower Income"
lending_club_df$annual_inccat[lending_club_df$annual_inc > 50000 & lending_club_df$an
nual_inc <= 100000] <- "Middle Income"
lending_club_df$annual_inccat[lending_club_df$annual_inc > 100000] <- "High Income"

#merging category in purpose
lending_club_df <- lending_club_df %>% mutate(purpose_merge = case_when(
                          purpose == "car" ~ "other",
                          purpose =="educational" ~ "other",
                          purpose =="home_improvement" ~ "other",
                          purpose =="house" ~ "other",
                          purpose =="major_purchase" ~ "other",
                          purpose =="medical" ~ "other",
                          purpose =="moving" ~ "other",
                          purpose =="small_business" ~ "other",
                          purpose =="vacation" ~ "other",
                          purpose =="wedding" ~ "other",
                          purpose =="renewable_energy" ~ "other",
                          purpose =="credit_card" ~ "credit_card",
                          purpose =="debt_consolidation" ~ "debt_consolidation",
                          purpose =="other" ~ "other"
                          ))
# merging sub-categories in emp_length
lending_club_df <- lending_club_df %>% mutate(emp_length_merge = case_when(
                          emp_length == "< 1 year" ~ "0 to 3 year",
                          emp_length =="1 year" ~ "0 to 3 year",
                          emp_length =="2 years" ~ "0 to 3 year",
                          emp_length =="3 years" ~ "0 to 3 year",
                          emp_length =="4 years" ~ "3 to 6",
                          emp_length =="5 years" ~ "3 to 6",
                          emp_length =="6 years" ~ "3 to 6",
                          emp_length =="7 years" ~ "6+ years",
                          emp_length =="8 years" ~ "6+ years",
                          emp_length =="9 years" ~ "6+ years",
                          emp_length =="10+ years" ~ "6+ years",
                          emp_length == "" ~ "Not Specified"
                          ))
#Factor the newly added columns
lending_club_df <- lending_club_df %>% mutate_at(vars(
                          annual_inccat,
                          purpose_merge,
                          emp_length_merge),
                          list(factor))

# Convert target variable into binary
names(lending_club_df)[names(lending_club_df) == "loan_status"] <-"default"
lending_club_df$default<-as.character(lending_club_df$default)
lending_club_df$default[lending_club_df$default == "Charged Off"] <- 1
lending_club_df$default[lending_club_df$default == "Fully Paid"] <- 0
lending_club_df$default<-as.factor(lending_club_df$default)
#Check data types
str(lending_club_df)
```

```
## 'data.frame':    395219 obs. of  31 variables:
##  $ loan_amnt           : num  10000 8000 15600 7200 24375 ...
##  $ term                : Factor w/ 2 levels " 36 months"," 60 months": 1 1 1 1 2 1 
1 1 2 1 ...
##  $ int_rate            : num  11.44 11.99 10.49 6.49 17.27 ...
##  $ installment         : num  329 266 507 221 609 ...
##  $ grade               : Factor w/ 7 levels "A","B","C","D",..: 2 2 2 1 3 3 1 2 2 
3 ...
##  $ sub_grade           : Factor w/ 35 levels "A1","A2","A3",..: 9 10 8 2 15 13 1 7 
8 15 ...
##  $ emp_title           : chr  "Marketing" "Credit analyst " "Statistician" "Client 
Advocate" ...
##  $ emp_length          : Factor w/ 12 levels "","< 1 year",..: 4 7 2 9 12 4 5 4 4 
6 ...
##  $ home_ownership      : Factor w/ 6 levels "ANY","MORTGAGE",..: 6 2 6 6 2 2 2 6 6 
2 ...
##  $ annual_inc          : num  117000 65000 43057 54000 55000 ...
##  $ verification_status : Factor w/ 3 levels "Not Verified",..: 1 1 2 1 3 3 2 1 3 3 
...
##  $ issue_d             : Factor w/ 115 levels "Apr-2008","Apr-2009",..: 46 46 46 9 
3 6 114 114 111 103 5 ...
##  $ default             : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1 1 ...
##  $ purpose             : Factor w/ 14 levels "car","credit_card",..: 13 3 2 2 2 3 
5 2 3 3 ...
##  $ title               : chr  "Vacation" "Debt consolidation" "Credit card refinan
cing" "Credit card refinancing" ...
##  $ dti                 : num  26.2 22.1 12.8 2.6 34 ...
##  $ earliest_cr_line    : Factor w/ 684 levels "Apr-1955","Apr-1958",..: 379 337 10
7 677 441 280 105 665 383 156 ...
##  $ open_acc            : num  16 17 13 6 13 8 8 11 13 13 ...
##  $ pub_rec             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ revol_bal           : num  36369 20131 11987 5472 24584 ...
##  $ revol_util          : num  41.8 53.3 92.2 21.5 69.8 ...
##  $ total_acc           : num  25 27 26 13 43 23 25 15 40 37 ...
##  $ initial_list_status : Factor w/ 2 levels "f","w": 2 1 1 1 1 1 1 1 2 1 ...
##  $ application_type    : Factor w/ 3 levels "DIRECT_PAY","INDIVIDUAL",..: 2 2 2 2 
2 2 2 2 2 2 ...
##  $ mort_acc            : num  0 3 0 0 1 4 3 0 3 1 ...
##  $ pub_rec_bankruptcies: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ address             : chr  "0174 Michelle Gateway\nMendozaberg, OK 22690" "1076 
Carney Fort Apt. 347\nLoganmouth, SD 05113" "87025 Mark Dale Apt. 269\nNew Sabrina, W
V 05113" "823 Reid Ford\nDelacruzside, MA 00813" ...
##  $ state               : Factor w/ 54 levels "AA","AE","AK",..: 40 45 53 23 49 12 
47 2 5 49 ...
##  $ annual_inccat       : Factor w/ 3 levels "High Income",..: 1 3 2 3 3 3 1 2 1 1 
...
##  $ purpose_merge       : Factor w/ 3 levels "credit_card",..: 3 2 1 1 1 2 3 1 2 2 
...
##  $ emp_length_merge    : Factor w/ 4 levels "0 to 3 year",..: 3 2 1 2 3 3 1 3 3 1 
...
```

Splitting train, Calibration and Test dataset

The data is then divided into three groups: Train data is used to train the model, whereas calibration data is used to validate the model for assessment, and finally, test data is used to validate the model once again.The proportions will be split into 80% train,10% calibration,10% test.

```
# Create random training, validation, and test sets
df <- lending_club_df
#spitting
fractionTraining   <- 0.80
fractionValidation <- 0.10
fractionTest       <- 0.10
# compute sample size
sampleSizeTraining   <- floor(fractionTraining   * nrow(df))
sampleSizeValidation <- floor(fractionValidation * nrow(df))
sampleSizeTest       <- floor(fractionTest       * nrow(df))
# Create the randomly-sampled indices for the dataframe. Use setdiff() to
#avoid overlapping subsets of indices
indicesTraining    <- sort(sample(seq_len(nrow(df)), size=sampleSizeTraining))
indicesNotTraining <- setdiff(seq_len(nrow(df)), indicesTraining)
indicesValidation  <- sort(sample(indicesNotTraining, size=sampleSizeValidation))
indicesTest        <- setdiff(indicesNotTraining, indicesValidation)

dTrain   <- df[indicesTraining, ]
dCal <- df[indicesValidation, ]
dTest      <- df[indicesTest, ]

dim(dTrain)
```

```
## [1] 316175     31
```

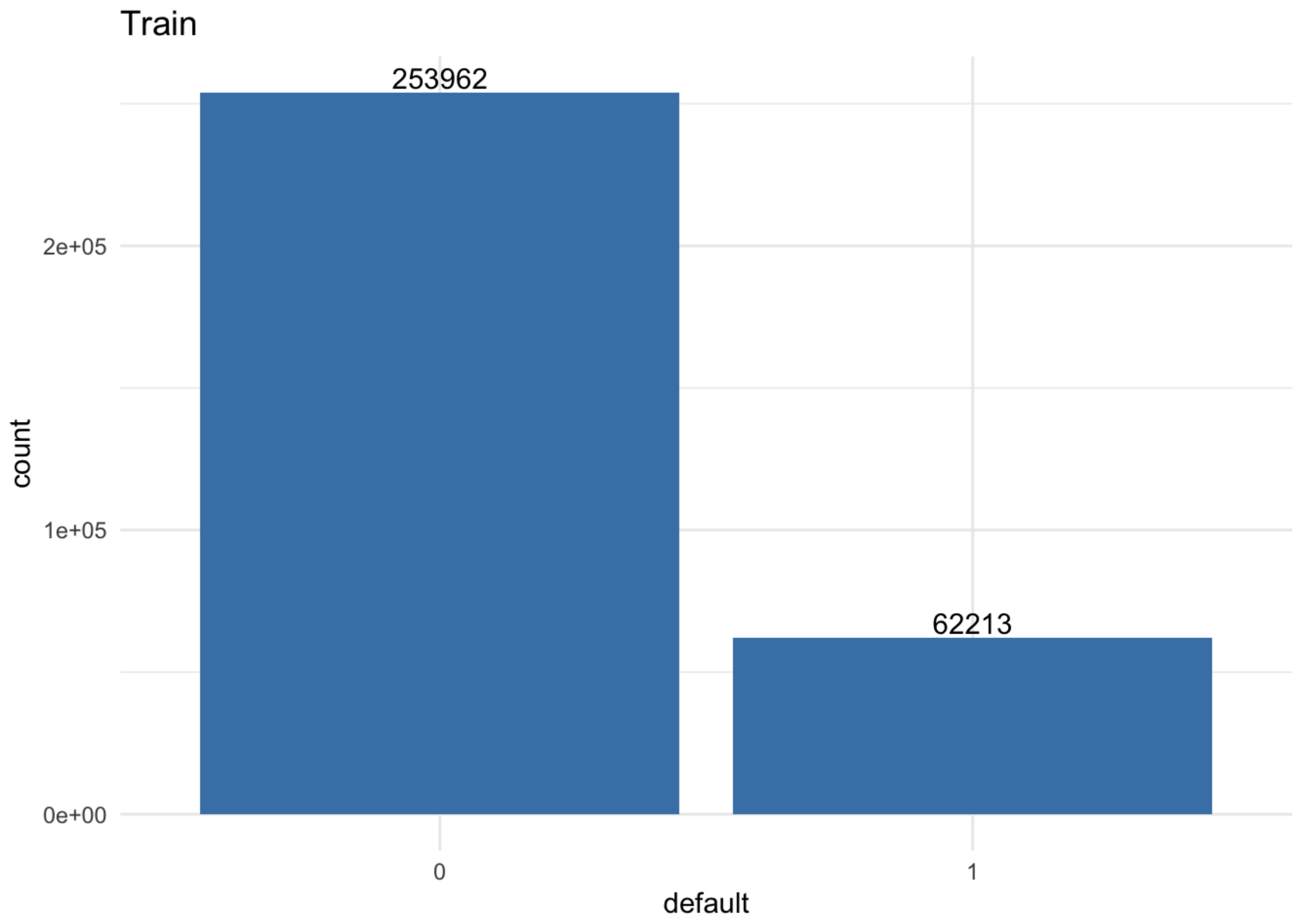
```
dim(dCal)
```

```
## [1] 39521    31
```

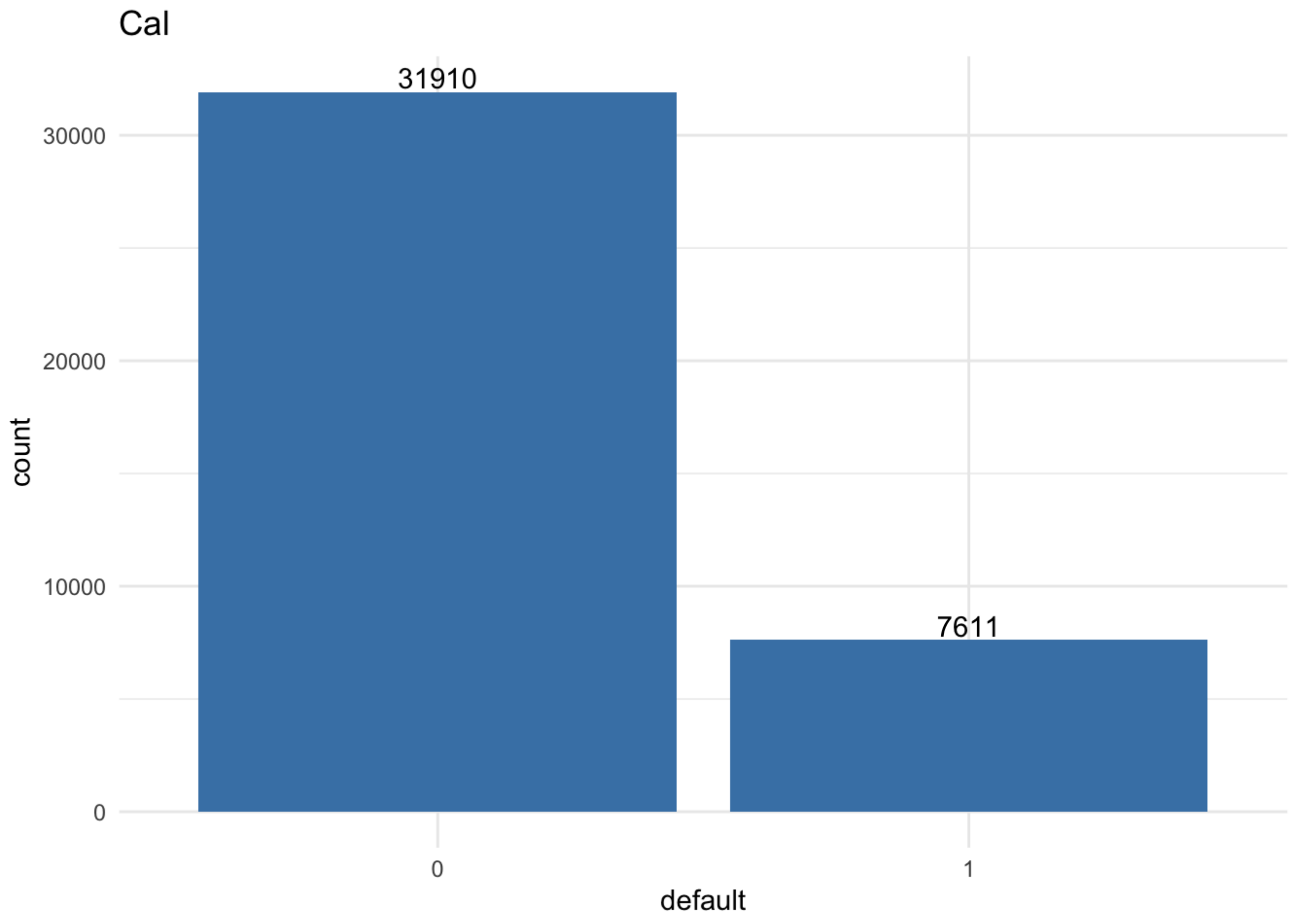
```
dim(dTest)
```

```
## [1] 39523    31
```

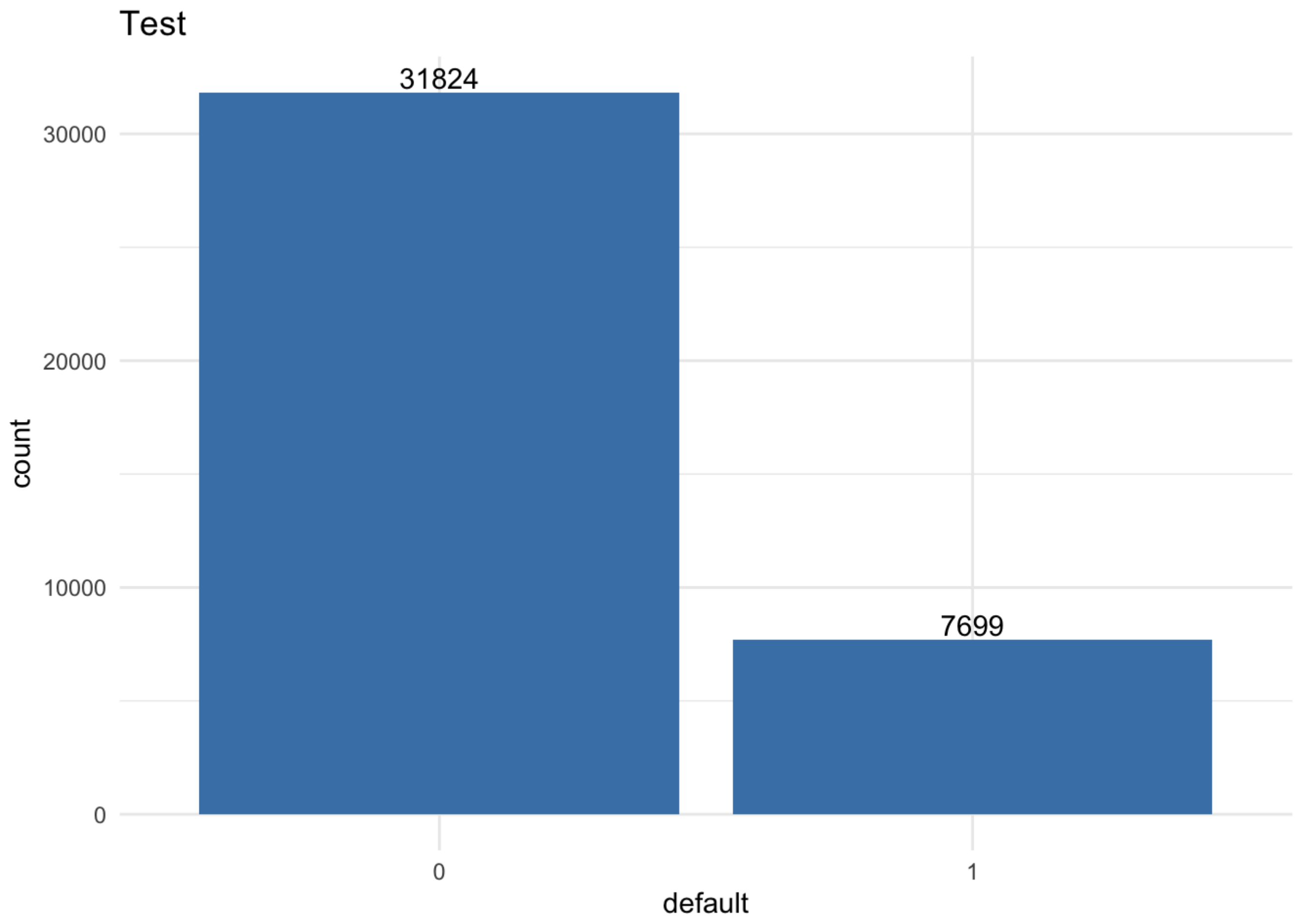
```
#checking class balance of target features on training set
#ratio of target variable is imbalanced
ggplot(dTrain, aes(x=default)) +
  geom_bar(fill="steelblue")+
  geom_text(stat = 'count',aes(label =..count..,
                               vjust = -0.2)) +
  theme_minimal() + ggtitle("Train")
```



```
ggplot(dCal, aes(x=default)) +
  geom_bar(fill="steelblue")+
  geom_text(stat = 'count',aes(label =..count..,
                             vjust = -0.2)) +
  theme_minimal() + ggtitle("Cal")
```



```
ggplot(dTest, aes(x=default)) +
  geom_bar(fill="steelblue")+
  geom_text(stat = 'count',aes(label =..count..,
                             vjust = -0.2)) +
  theme_minimal() +ggtitle("Test")
```



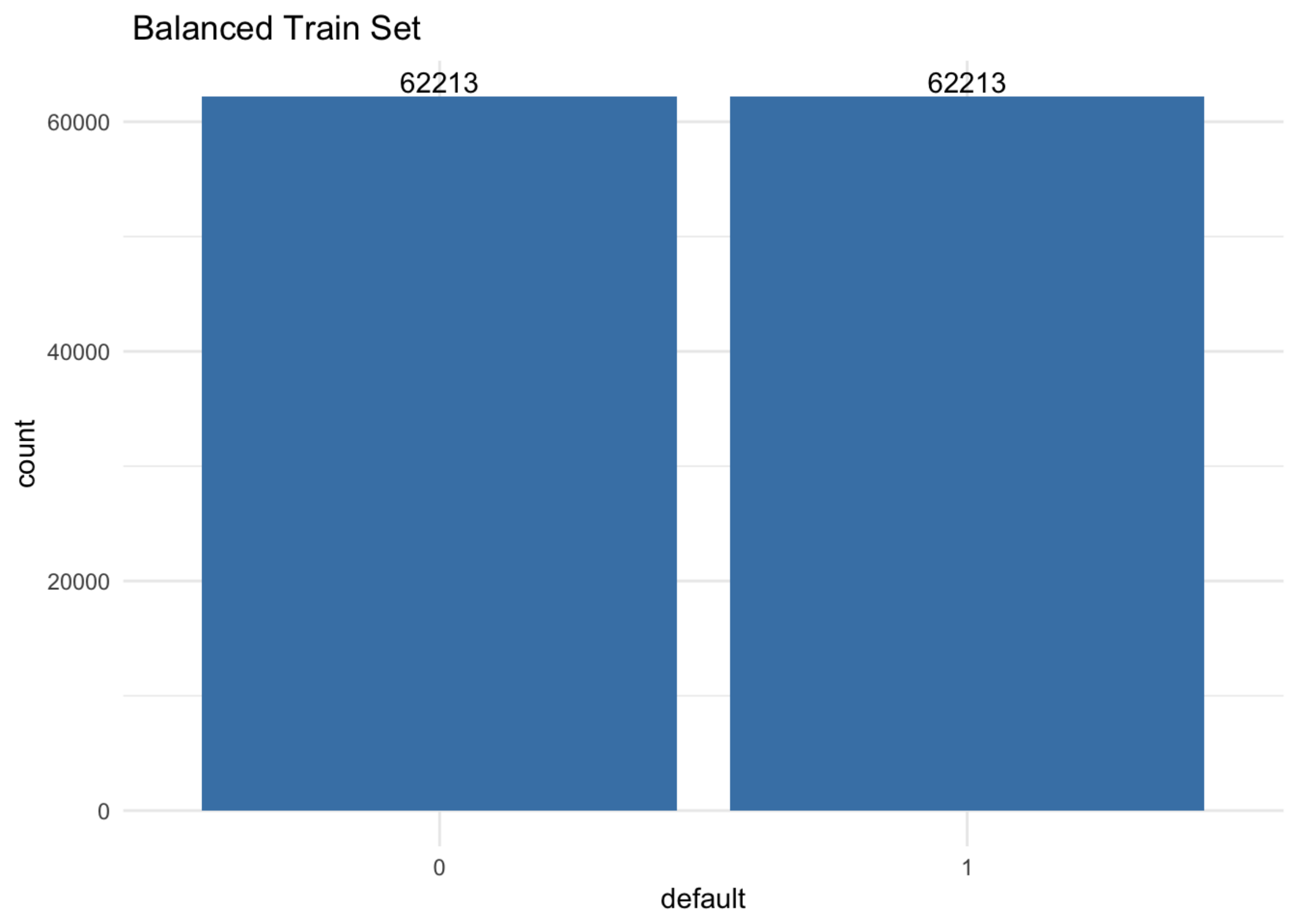
Class Imbalanced

Undersampling is a technique to balance uneven data-sets by keeping all of the data in the minority class and decreasing the size of the majority class.

```
#Undersampling is to correct imbalanced data to reduce the risk of their analysis or
machine learning algorithm skewing toward the majority.
train_down <-
  caret::downSample(x = dTrain[, !(names(dTrain) %in% c("default"))], y = as.factor(d
Train$default), yname = "default")

base::prop.table(table(train_down$default))
```

```
##
##   0   1
## 0.5 0.5
```

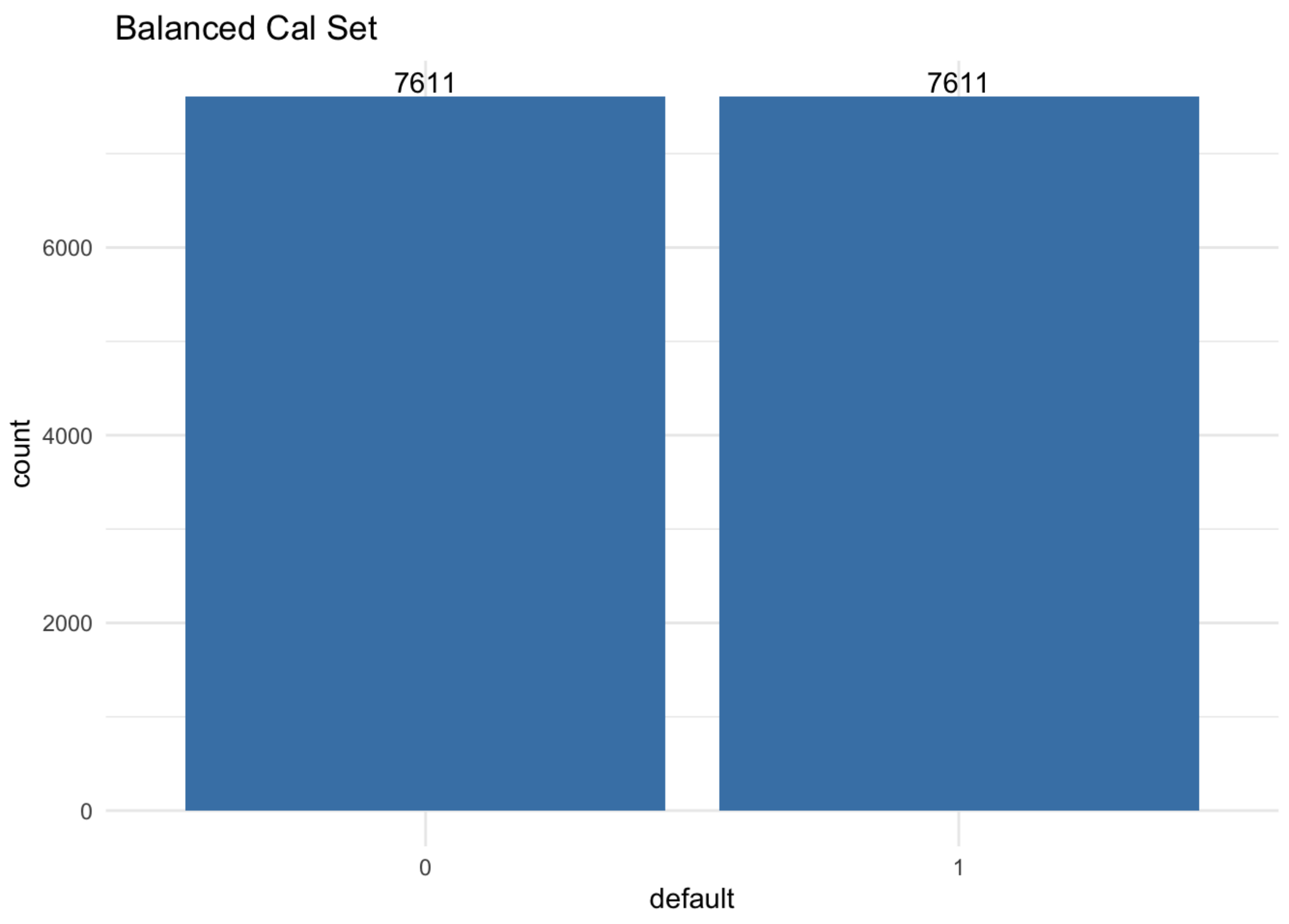
```
ggplot(train_down, aes(x=default)) +
  geom_bar(fill="steelblue")+
  geom_text(stat = 'count',aes(label =..count..,
                              vjust = -0.2)) +
  theme_minimal() + ggtitle(" Balanced Train Set")
```



```
cal_down <-
  caret::downSample(x = dCal[, !(names(dCal) %in% c("default"))], y = as.factor(dCal
$default), yname = "default")

base::prop.table(table(cal_down$default))
```

```
##
##   0   1
## 0.5 0.5
```

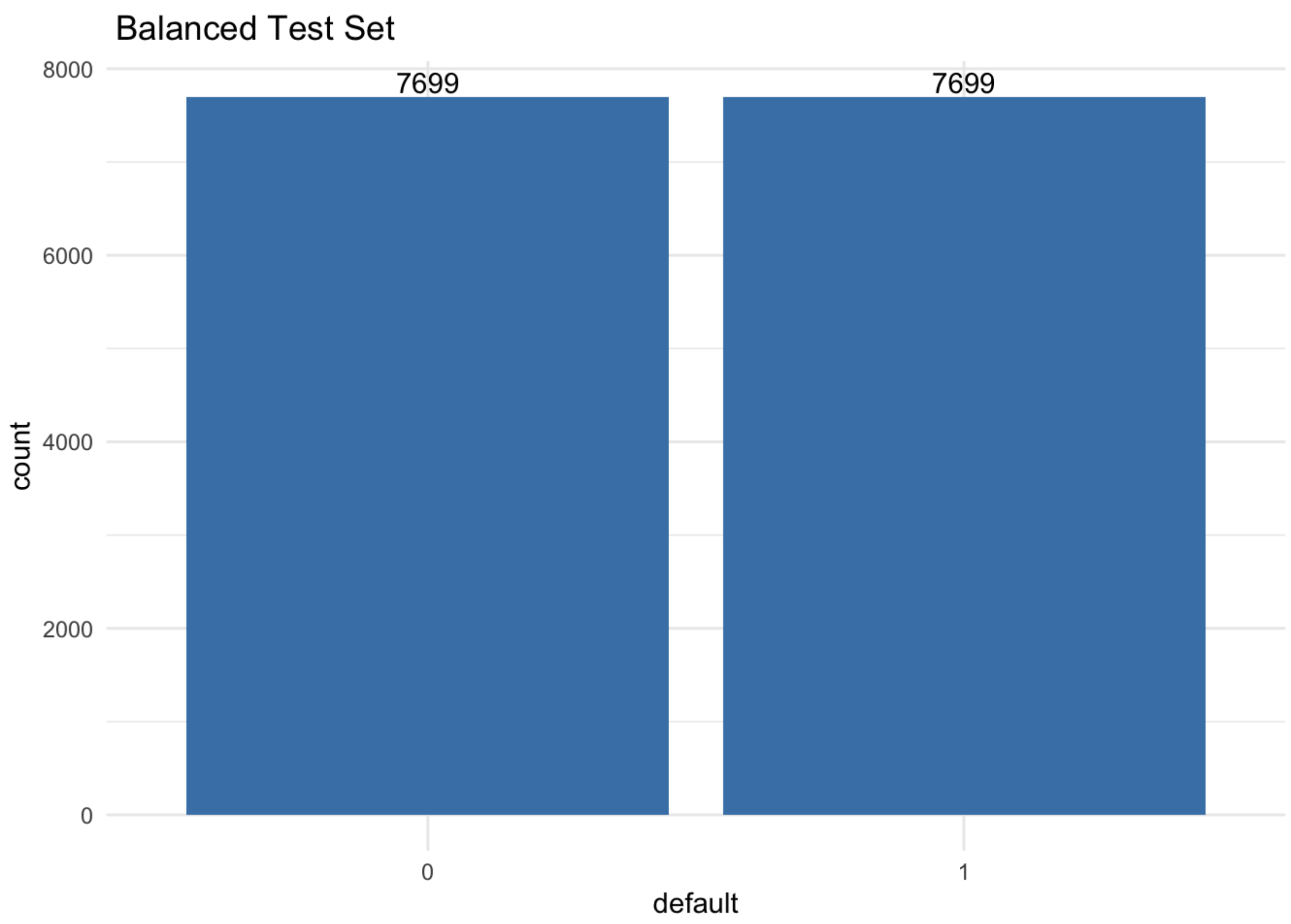
```
ggplot(cal_down, aes(x=default)) +
  geom_bar(fill="steelblue")+
  geom_text(stat = 'count',aes(label =..count..,
                              vjust = -0.2)) +
  theme_minimal() + ggtitle(" Balanced Cal Set")
```



```
test_down <-
  caret::downSample(x = dTest[, !(names(dTest) %in% c("default"))], y = as.factor(dTe
st$default), yname = "default")

base::prop.table(table(test_down$default))
```

```
##
##   0   1
## 0.5 0.5
```

```
ggplot(test_down, aes(x=default)) +
  geom_bar(fill="steelblue")+
  geom_text(stat = 'count',aes(label =..count..,
                             vjust = -0.2)) +
  theme_minimal() + ggtitle(" Balanced Test Set")
```



```
# removing all dataset replacing with the new balance set.
remove(dCal, dTest, dTrain)
dTrain<-train_down
dTest<-test_down
dCal<-cal_down
```

Data Modelling

1. NULL Model Null models are useful for determining the performance's lowest bound — We're not producing value if you don't outperform the null model.

```
pos<- "1"
outcome <- "default"
number_of_charged_off<-sum(dTrain[,outcome] == pos)
dCal$null.prediction<- number_of_charged_off / nrow(dTrain)
TP <- 0
TN <- sum(dCal[,outcome] == "0"); # using threshold 0.5
FP <- 0
FN <- sum(dCal[,outcome] == "1"); # using threshold 0.5
cat("nrow(dCal):", nrow(dCal), "TP:", TP, "TN:", TN, "FP:", FP, "FN:", FN)
```

```
## nrow(dCal): 15222 TP: 0 TN: 7611 FP: 0 FN: 7611
```

```
(accuracy <- (TP + TN) / nrow(dCal))
```

```
## [1] 0.5
```

```
(precision <- TP/(TP + FP))
```

```
## [1] NaN
```

```
(recall <- TP/(TP + FN))
```

```
## [1] 0
```

```
#Model evaluation
calcAUC <- function(predcol,outcol) {
perf <- ROCR::performance(prediction(predcol,outcol==pos),'auc')
as.numeric(perf@y.values) }
AUC <- calcAUC(dCal$null.prediction, dCal[,outcome])
```

2. Single Variable Model

```
#Function for Single Variable Model: Categorical variables
outcome <- 'default'
pos <- "1"
vars <- c( "term", "int_rate", "installment", "grade", "emp_length_merge", "home_owne
rship", "purpose_merge",  "verification_status", "dti", "open_acc", "pub_rec", "revol
_bal", "revol_util", "total_acc", "initial_list_status", "application_type", "mort_ac
c", "pub_rec_bankruptcies")
catVars <- vars[sapply(dTrain[, vars], class) %in% c('factor', 'character')]
# #Function for Single Variable Model: Categorical variables
mkPredC <- function(outCol, varCol, appCol) {
  pPos <- sum(outCol == pos) / length(outCol)
  naTab <- table(as.factor(outCol[is.na(varCol)]))
  pPosWna <- (naTab/sum(naTab))[pos]
  vTab <- table(as.factor(outCol), varCol)
  pPosWv <- (vTab[pos, ] + 1.0e-3*pPos) / (colSums(vTab) + 1.0e-3)
  pred <- pPosWv[appCol]
  pred[is.na(appCol)] <- pPosWna
  pred[is.na(pred)] <- pPos
  pred
}

#Predict using the Single Variable Model
for(v in catVars) {
  pi <- paste('pred', v, sep='_')
  dTrain[,pi] <- mkPredC(dTrain[,outcome], dTrain[,v], dTrain[,v])
  dCal[,pi] <- mkPredC(dTrain[,outcome], dTrain[,v], dCal[,v])
  dTest[,pi] <- mkPredC(dTrain[,outcome], dTrain[,v], dTest[,v])
}

for(v in catVars) {
  pi <- paste('pred', v, sep='_')
  aucTrain <- calcAUC(dTrain[,pi], dTrain[,outcome])
  if (aucTrain >= 0.51) {
    aucCal <- calcAUC(dCal[,pi], dCal[,outcome])
    print(sprintf(
      "%s: trainAUC: %4.3f; calibrationAUC: %4.3f",
      pi, aucTrain, aucCal))
  }
}
```

```
## [1] "pred_term: trainAUC: 0.592; calibrationAUC: 0.595"
## [1] "pred_grade: trainAUC: 0.677; calibrationAUC: 0.679"
## [1] "pred_emp_length_merge: trainAUC: 0.516; calibrationAUC: 0.515"
## [1] "pred_home_ownership: trainAUC: 0.544; calibrationAUC: 0.548"
## [1] "pred_purpose_merge: trainAUC: 0.524; calibrationAUC: 0.529"
## [1] "pred_verification_status: trainAUC: 0.554; calibrationAUC: 0.554"
```

```
vars <- c("term", "grade", "emp_length_merge", "home_ownership", "purpose_merge", "ve
rification_status")
for (var in vars) {
  aucs <- rep(0,100)
  for (rep in 1:length(aucs)) {
    useForCalRep <- rbinom(n=nrow(dTrain), size=1, prob=0.1) > 0
    predRep <- mkPredC(dTrain[!useForCalRep, outcome], dTrain[!useForCalRep, var],dTr
ain[useForCalRep, var])
    aucs[rep] <- calcAUC(predRep, dTrain[useForCalRep, outcome])
  }
  print(sprintf("%s: mean: %4.3f; sd: %4.3f", var, mean(aucs), sd(aucs)))
}
```

```
## [1] "term: mean: 0.592; sd: 0.004"
## [1] "grade: mean: 0.677; sd: 0.004"
## [1] "emp_length_merge: mean: 0.515; sd: 0.004"
## [1] "home_ownership: mean: 0.544; sd: 0.004"
## [1] "purpose_merge: mean: 0.524; sd: 0.004"
## [1] "verification_status: mean: 0.553; sd: 0.005"
```

```
#Model evaluation for the single-variable model: numerical
outcome <- 'default'
pos <- '1'
vars <- c( "term","annual_inc", "int_rate", "installment", "grade", "emp_length_merg
e", "home_ownership", "purpose_merge",  "verification_status", "dti", "open_acc", "pu
b_rec", "revol_bal", "revol_util", "total_acc", "initial_list_status", "application_t
ype", "mort_acc", "pub_rec_bankruptcies")
numericVars <- vars[sapply(dTrain[, vars], class) %in%
c('numeric', 'integer')]
mkPredN <- function(outCol, varCol, appCol) {
# compute the cuts
cuts <- unique(
quantile(varCol, probs=seq(0, 1, 0.1), na.rm=T))
# discretize the numerical columns
varC <- cut(varCol,cuts)
appC <- cut(appCol,cuts)
mkPredC(outCol,varC,appC)
}
for(v in numericVars) {
pi <- paste('pred', v, sep='_')
dTrain[,pi] <- mkPredN(dTrain[,outcome], dTrain[,v], dTrain[,v])
dCal[,pi] <- mkPredN(dTrain[,outcome], dTrain[,v], dCal[,v])
dTest[,pi] <- mkPredN(dTrain[,outcome], dTrain[,v], dTest[,v])
aucTrain <- calcAUC(dTrain[,pi], dTrain[,outcome])
if(aucTrain >= 0.55) {
aucCal <- calcAUC(dCal[,pi], dCal[,outcome])
print(sprintf(
"%s: trainAUC: %4.3f; calibrationAUC: %4.3f",
pi, aucTrain, aucCal))}}
```

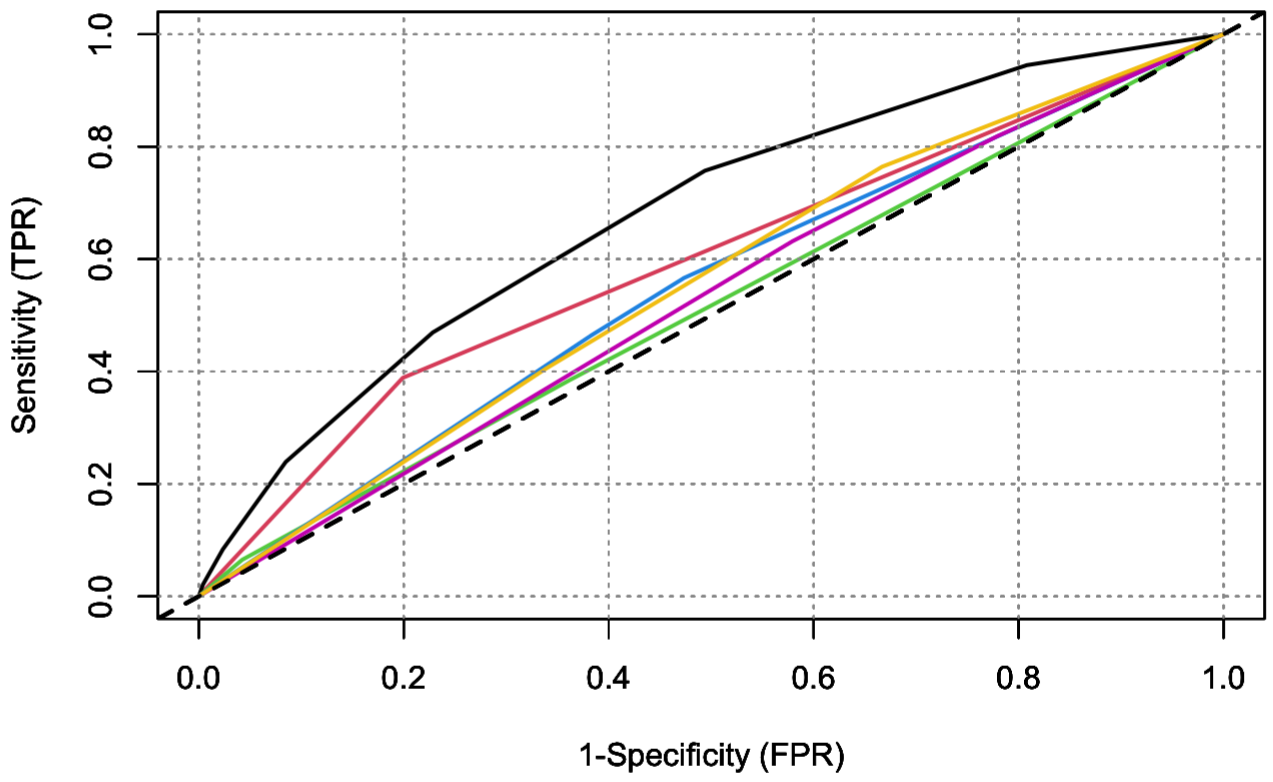
```
## [1] "pred_annual_inc: trainAUC: 0.561; calibrationAUC: 0.561"
## [1] "pred_int_rate: trainAUC: 0.675; calibrationAUC: 0.676"
## [1] "pred_dti: trainAUC: 0.591; calibrationAUC: 0.595"
## [1] "pred_revol_util: trainAUC: 0.559; calibrationAUC: 0.552"
## [1] "pred_mort_acc: trainAUC: 0.560; calibrationAUC: 0.566"
```

```
vars <- c('annual_inc', 'int_rate', 'dti','revol_util','mort_acc')
for (var in vars) {
aucs <- rep(0,100)
for (rep in 1:length(aucs)) {
useForCalRep <- rbinom(n=nrow(dTrain), size=1, prob=0.1) > 0
predRep <- mkPredN(dTrain[!useForCalRep, outcome],
dTrain[!useForCalRep, var],
dTrain[useForCalRep, var])
aucs[rep] <- calcAUC(predRep, dTrain[useForCalRep, outcome])
}
print(sprintf("%s: mean: %4.3f; sd: %4.3f", var, mean(aucs), sd(aucs)))
}
```

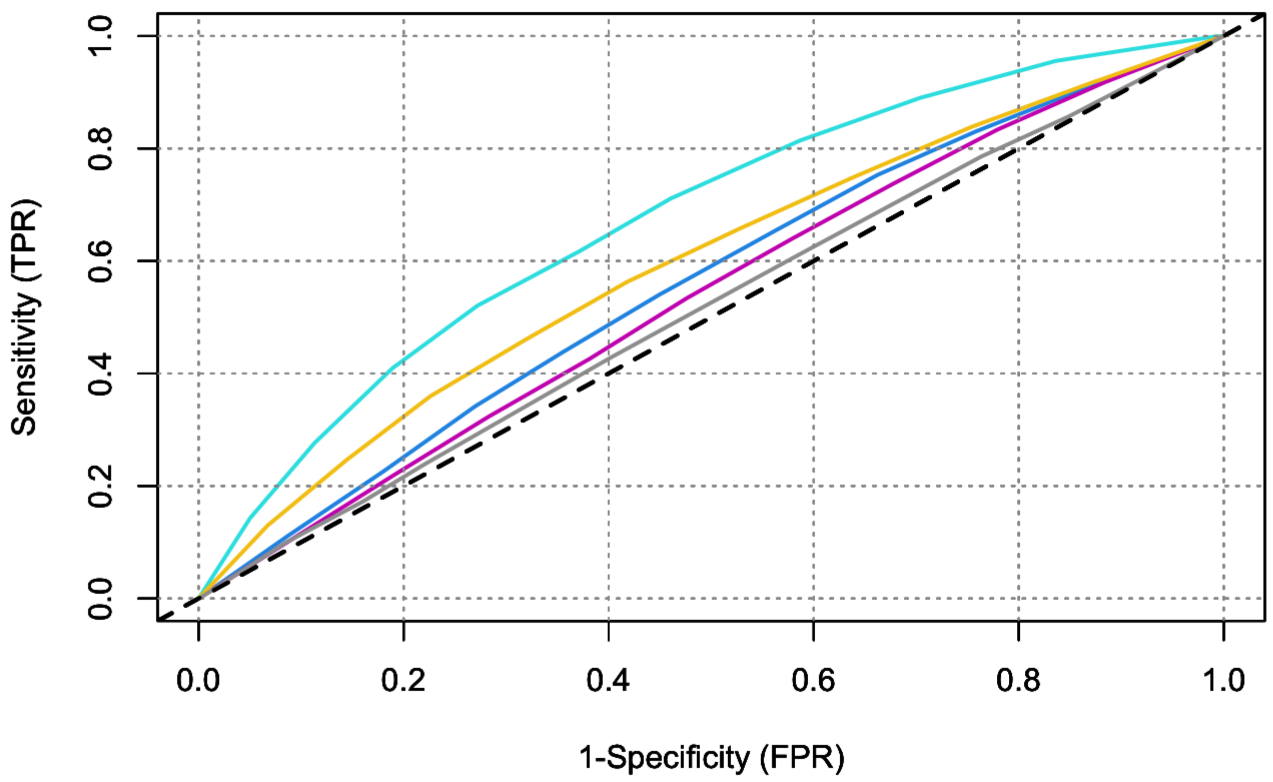
```
## [1] "annual_inc: mean: 0.560; sd: 0.005"
## [1] "int_rate: mean: 0.675; sd: 0.004"
## [1] "dti: mean: 0.592; sd: 0.005"
## [1] "revol_util: mean: 0.558; sd: 0.005"
## [1] "mort_acc: mean: 0.560; sd: 0.004"
```

The genuine positive rate (Sensitivity) is displayed as a function of the false positive rate (100-Specificity) at various cut-off points in a Receiver Operating Characteristic (ROC) curve.A sensitivity/specificity pair corresponding to a specific decision threshold is represented by each point on the ROC curve. The ROC curve of a test with perfect discrimination (no overlap between the two distributions) goes through the upper left corner (100 percent sensitivity, 100 percent specificity). As a result, the closer the ROC curve is to the upper left corner, the better the test's overall accuracy (Zweig & Campbell, 1993). An AUC of 0.5 indicates that the test has no discriminating capacity (i.e., it is no better than chance), whereas an AUC of 1.0 indicates that the test has perfect discrimination.

```
#Plotting the UACs from Single Variable Model to compare : Categorical
plot_roc_cat<- function(predcol, outcol, colour_id=2, overlaid=F) {
ROCit_obj <- rocit(score=predcol, class=outcol==pos)
par(new=overlaid)
plot(ROCit_obj, col = c(colour_id, 1),
legend = FALSE, YIndex = FALSE, values = FALSE)
}
plot_roc_cat(dCal$pred_term, dCal[,outcome])# red
plot_roc_cat(dCal$pred_grade, dCal[,outcome], colour_id = 1, overlaid = TRUE)# black
plot_roc_cat(dCal$pred_emp_length_merge, dCal[,outcome], colour_id = 3, overlaid = TR
UE)# green
plot_roc_cat(dCal$pred_home_ownership, dCal[,outcome],colour_id = 4, overlaid = TRUE)
# blue
plot_roc_cat(dCal$pred_purpose_merge, dCal[,outcome],colour_id = 6, overlaid = TRUE)#
purple
plot_roc_cat(dCal$pred_verification_status, dCal[,outcome],colour_id = 7, overlaid =
TRUE)# yellow
```



```
#Plotting the UACs from Single Variable Model to compare : Numerical
plot_roc_num<- function(predcol, outcol, colour_id=2, overlaid=F) {
ROCit_obj <- rocit(score=predcol, class=outcol==pos)
par(new=overlaid)
plot(ROCit_obj, col = c(colour_id, 1),
legend = FALSE, YIndex = FALSE, values = FALSE)
}
plot_roc_num(dCal$pred_annual_inc, dCal[,outcome], colour_id = 12, overlaid = TRUE)#
blue
plot_roc_num(dCal$pred_int_rate, dCal[,outcome], colour_id = 13, overlaid = TRUE)# li
ght blue
plot_roc_num(dCal$pred_installment, dCal[,outcome], colour_id = 14, overlaid = TRUE)#
purple
plot_roc_num(dCal$pred_dti, dCal[,outcome], colour_id = 15, overlaid = TRUE)# yellow
plot_roc_num(dCal$pred_open_acc, dCal[,outcome], colour_id = 16, overlaid = TRUE)# gr
ey
```



From the the list of single variables, it can be seen that Grade and Annual Income has AUC (Train, Calibration, Test) above 0.60.

FEATURE SELECTION

In developing a predictive model, feature selection is performed to reduce the input variables. Ideally, this process should be done in a way to minimize the input variables in order to minimize the computational cost of modeling, as well as improve the model's performance when possible. By using statistical analyses, you can select the input variables with the strongest relationships to the target variable using statistical methods. These methods may be fast and effective, though the type of statistical measure is dependent on the data type of both input and output variables. Based on deviance, we select features. To calculate deviance, the log likelihood of the predicted values from the model that accounts for one variable must be computed.

```
#Function to compute log likelihood.
logLikelihood <- function(outCol,predCol) {

      sum(ifelse(outCol==pos,log(predCol),log(1-predCol)))
}

#Compute the base rate of a NULL model
baseRateCheck <- logLikelihood(
                   dCal[,outcome],
                    sum(dCal[,outcome]==pos)/
                    length(dCal[,outcome]))



deviance_table <- data.frame()
#Category Feature
for (v in catVars) {
    pi <- paste('pred',v,sep="_")
    liCheck <- 2*((logLikelihood(dCal[,outcome],dCal[,pi]) - baseRateCheck))
    print(sprintf("%s, calibrationScore: %g",pi,liCheck))
deviance_temp=data.frame(predictors=pi,deviance=liCheck)
deviance_table=rbind(deviance_table,deviance_temp)
}
```

```
## [1] "pred_term, calibrationScore: 667.609"
## [1] "pred_grade, calibrationScore: 1610.06"
## [1] "pred_emp_length_merge, calibrationScore: 39.3185"
## [1] "pred_home_ownership, calibrationScore: 130.589"
## [1] "pred_purpose_merge, calibrationScore: 46.4056"
## [1] "pred_verification_status, calibrationScore: 183.026"
## [1] "pred_initial_list_status, calibrationScore: 4.89759"
## [1] "pred_application_type, calibrationScore: -1.12117"
```

```
deviance_table <- deviance_table[order(-deviance_table$deviance),]
deviance_table
```

```
##                  predictors    deviance
## 2                pred_grade 1610.063053
## 1                 pred_term  667.608982
## 6 pred_verification_status  183.026305
## 4      pred_home_ownership  130.588558
## 5       pred_purpose_merge   46.405554
## 3    pred_emp_length_merge   39.318530
## 7 pred_initial_list_status    4.897585
## 8    pred_application_type   -1.121174
```

```r
deviance_table <- data.frame()
for(v in numericVars) {
 pii <- paste('pred',v,sep='_')
  liCheck <- 2*((logLikelihood(dCal[,outcome],dCal[,pii])
                 - baseRateCheck) - 1)
    print(sprintf("%s, calibrationScore: %g", pii,liCheck))
deviance_temp=data.frame(predictors=pii,deviance=liCheck)
deviance_table=rbind(deviance_table,deviance_temp)

  }
```

```
## [1] "pred_annual_inc, calibrationScore: 172.438"
## [1] "pred_int_rate, calibrationScore: 1512.99"
## [1] "pred_installment, calibrationScore: 84.325"
## [1] "pred_dti, calibrationScore: 425.563"
## [1] "pred_open_acc, calibrationScore: 12.8693"
## [1] "pred_pub_rec, calibrationScore: 11.4745"
## [1] "pred_revol_bal, calibrationScore: 3.71242"
## [1] "pred_revol_util, calibrationScore: 132.77"
## [1] "pred_total_acc, calibrationScore: 7.40308"
## [1] "pred_mort_acc, calibrationScore: 210.591"
## [1] "pred_pub_rec_bankruptcies, calibrationScore: -1.04785"
```

```r
pos <- '1'
# Define a function to compute log likelihood so that we can reuse it.
logLikelihood <- function(ytrue, ypred) {
sum(ifelse(ytrue==pos, log(ypred), log(1-ypred)), na.rm=T)
}
# Compute the likelihood of the Null model on the calibration
# set (for the KDD dataset from previous lecture)
outcome <- 'default'
logNull <- logLikelihood(
dCal[,outcome], sum(dCal[,outcome]==pos)/nrow(dCal)
)
cat(logNull)
```

```
## -10551.09
```

```r
selCatVars <- c()
minDrop <- 100
for (v in catVars) {
  pi <- paste('pred',v,sep="_")
  drop <- 2*(logLikelihood(dCal[,outcome],dCal[,pi]) - logNull)
  if (drop >= minDrop) {
    print(sprintf("%s, deviance reduction: %g", pi, drop))
    selCatVars <- c(selCatVars, pi)}
}
```

```
## [1] "pred_term, deviance reduction: 667.609"
## [1] "pred_grade, deviance reduction: 1610.06"
## [1] "pred_home_ownership, deviance reduction: 130.589"
## [1] "pred_verification_status, deviance reduction: 183.026"
```

```
selNumVars <- c()
minDrop <- 100
for (v in numericVars) {
pi <- paste('pred', v, sep='_')
devDrop <- 2*(logLikelihood(dCal[,outcome], dCal[,pi]) - logNull)
if (devDrop >= minDrop) {
print(sprintf("%s, deviance reduction: %g", pi, devDrop))
selNumVars <- c(selNumVars, pi)
}
}
```

```
## [1] "pred_annual_inc, deviance reduction: 174.438"
## [1] "pred_int_rate, deviance reduction: 1514.99"
## [1] "pred_dti, deviance reduction: 427.563"
## [1] "pred_revol_util, deviance reduction: 134.77"
## [1] "pred_mort_acc, deviance reduction: 212.591"
```

```
(selVars <- c(selCatVars, selNumVars))
```

```
## [1] "pred_term"                "pred_grade"
## [3] "pred_home_ownership"      "pred_verification_status"
## [5] "pred_annual_inc"          "pred_int_rate"
## [7] "pred_dti"                 "pred_revol_util"
## [9] "pred_mort_acc"
```

Based on the predictors vs. deviance above, there seems to be quite a few features that we can use.

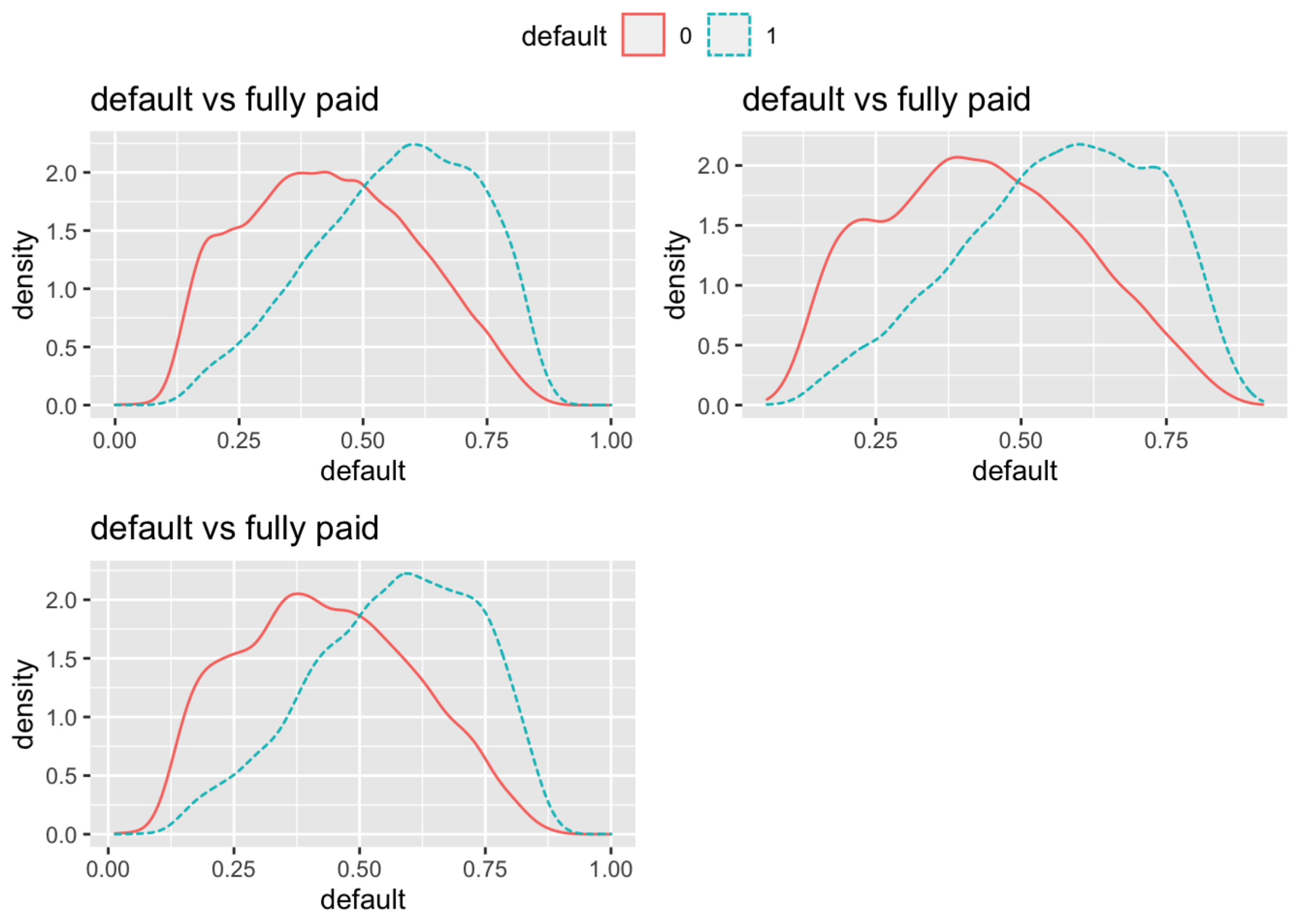
3. Logistic Regression Model

```
#all variable
fV <- paste(outcome,' ~ ', paste(c(catVars,numericVars), collapse=' + '), sep='')
model_logr <- glm(formula=fV, data=dTrain, family=binomial(link="logit"))
dTrain$pred <- predict(model_logr, newdata=dTrain, type="response")
dCal$pred <- predict(model_logr, newdata=dCal, type="response")
dTest$pred <- predict(model_logr, newdata=dTest, type="response")
```

```
  glm.aucTrain <- calcAUC(dTrain$pred,dTrain[,outcome])
  glm.aucCal<- calcAUC(dCal$pred,dCal[,outcome])
  glm.aucTest<- calcAUC(dTest$pred,dTest[,outcome])

  print(sprintf("glm.trainAUC: %4.3f , glm.calAUC: %4.3f , glm.testAUC: %4.3f",
                glm.aucTrain, glm.aucCal, glm.aucTest))
```

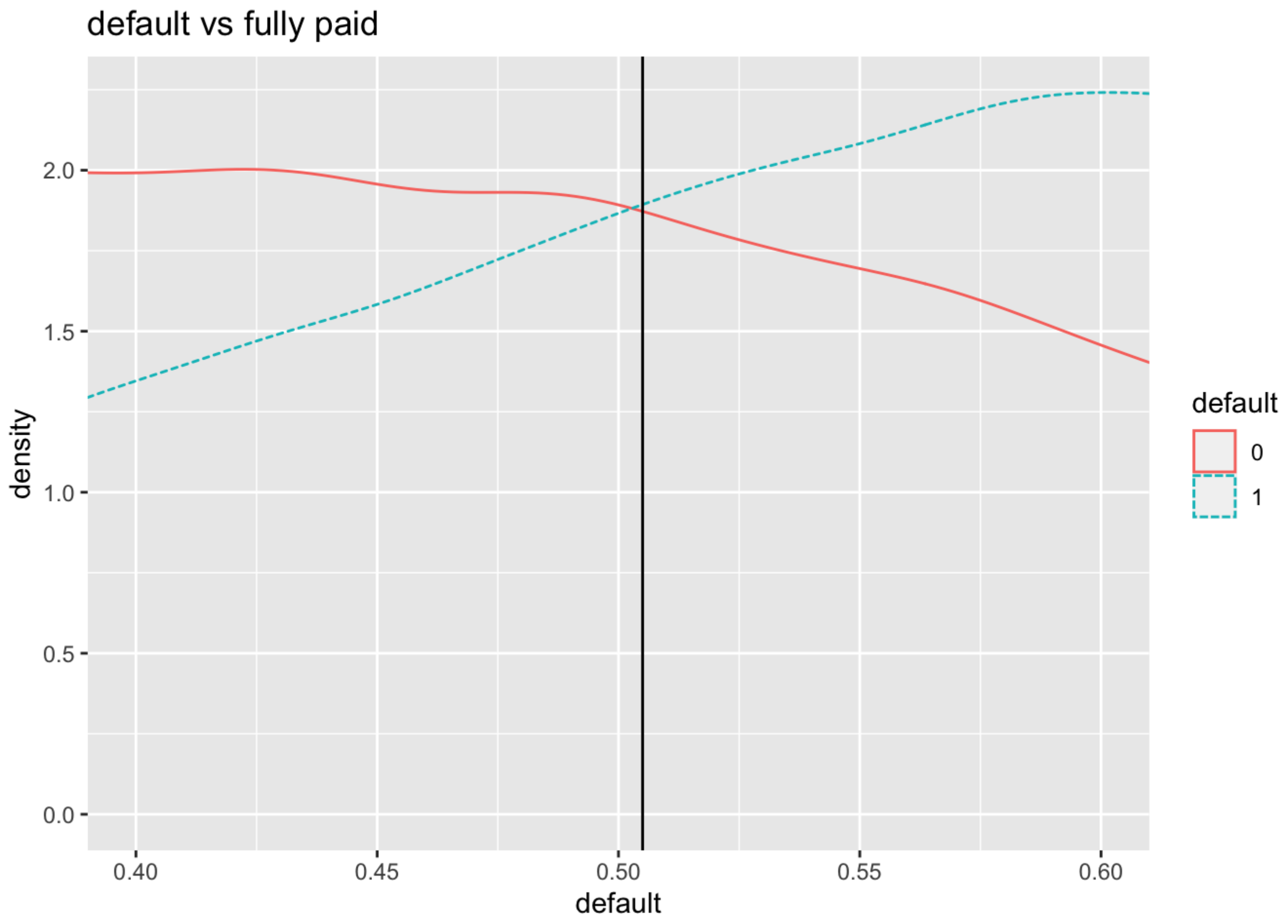
```
## [1] "glm.trainAUC: 0.707 , glm.calAUC: 0.710 , glm.testAUC: 0.711"
```

```
#Density plot to determine the likelihood treshold
train_glm_des<- ggplot(dTrain,aes(x=pred,color=default,linetype=default))+geom_densit
y()+
  labs(title="default vs fully paid",x="default",
       linetype="default",colour="default")

cal_glm_des<- ggplot(dCal,aes(x=pred,color=default,linetype=default))+geom_density()+
  labs(title="default vs fully paid",x="default",
       linetype="default",colour="default")

test_glm_des<- ggplot(dTest,aes(x=pred,color=default,linetype=default))+geom_density
()+
  labs(title="default vs fully paid",x="default",
       linetype="default",colour="default")

ggarrange(train_glm_des,cal_glm_des,test_glm_des,
          font.label=list(size=5),
          vjust=0.5,
          ncol=2,nrow=2,
          common.legend = TRUE)
```



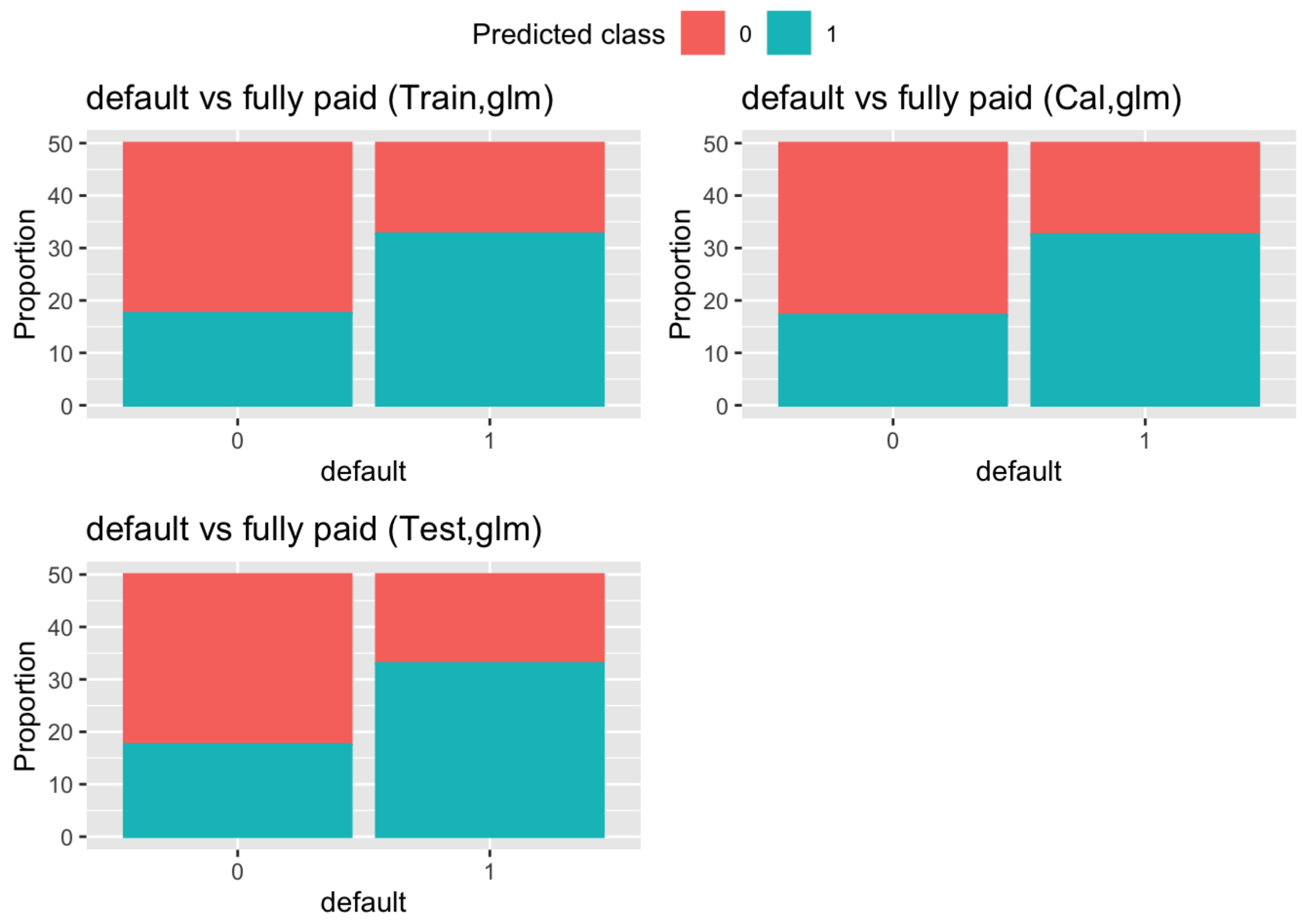
Based on the graph above, the density curves all 3 datasets intersect at one point. Next, the density curves for train data will be zoomed in to look at the intersection point.

```
train_glm_des+ coord_cartesian(xlim = c(0.40, 0.60))+geom_vline(xintercept=0.505)
```



Based on the graph generated from train data, the intersection point is at 0.5. Therefore, if the predicted probability value is more than 0.5. then the applicant will be under default. Else, the applicant will be under non-default. Next, the proportion of predicted Streams versus actual Streams will be visualise using bar chart to look at the proportion of correct classification.

```
glm_train_df <- data.frame(Actual_cls = dTrain$default,
                           Pred_cls = as.factor(ifelse
                                    (dTrain$pred>0.505,"1","0")))
glm_train_tab <- data.frame(with(glm_train_df,prop.table(table(Actual_cls,Pred_cl
s))))


glm_cal_df <- data.frame(Actual_cls=dCal$default,
                        Pred_cls=
                          as.factor (ifelse(dCal$pred>0.505,"1","0")))

glm_cal_tab <- data.frame(with(glm_cal_df,prop.table(table(Actual_cls,Pred_cls))))

glm_test_df <- data.frame(Actual_cls=dTest$default,
                        Pred_cls=
                          as.factor(ifelse(dTest$pred>0.505,"1","0")))

glm_test_tab <- data.frame(with(glm_test_df,prop.table(table(Actual_cls,Pred_cls))))

train_glm <- ggplot(data=glm_train_tab,aes(x=Actual_cls,y=Freq*100,group=Pred_cls,
                              colour=Pred_cls,fill=Pred_cls))+
  geom_col()+
  labs(title="default vs fully paid (Train,glm)",x="default",y="Proportion",
       fill="Predicted class",colour="Predicted class")

cal_glm<- ggplot(data=glm_cal_tab,aes(x=Actual_cls,y=Freq*100,group=Pred_cls,
                              colour=Pred_cls,fill=Pred_cls))+
  geom_col()+
  labs(title="default vs fully paid (Cal,glm)",x="default",y="Proportion",
       fill="Predicted class",colour="Predicted class")

test_glm<- ggplot(data=glm_test_tab,aes(x=Actual_cls,y=Freq*100,group=Pred_cls,
                              colour=Pred_cls,fill=Pred_cls))+
  geom_col()+
  labs(title="default vs fully paid (Test,glm)",x="default",y="Proportion",
       fill="Predicted class",colour="Predicted class")

ggarrange(train_glm,cal_glm,test_glm,
          font.label=list(size=3),
          vjust=0.5,
          ncol=2,nrow=2,
          common.legend = TRUE)
```



Based on the charts above, logistic regression seem to be better in identifying both applicants. Next, confusion matrix is calculated to look at recall and specificity of the model for each dataset.

```
# confusion matrix logistics
cf_glm_train <- confusionMatrix(glm_train_df[,2],glm_train_df[,1],
                             positive="1",mode="everything")
cf_glm_cal <- confusionMatrix(glm_cal_df[,2],glm_cal_df[,1],
                           positive="1",mode="everything")
cf_glm_test <- confusionMatrix(glm_test_df[,2],glm_test_df[,1],
                            positive="1",mode="everything")

cf_glm <- list(cf_glm_train,cf_glm_cal,cf_glm_test)

sp_glm <- data.frame()
dataset_name <- c("Train","Calibration","Test")

for(i in c(1:length(cf_glm))){
  sp_glm_temp <- data.frame(dataset=dataset_name[i],Recall=cf_glm[[i]]$byClass["Recal
l"],Specificity=cf_glm[[i]]$byClass["Specificity"])
  sp_glm <- rbind(sp_glm,sp_glm_temp)
}

sp_glm
```

```
##             dataset    Recall Specificity
## Recall        Train 0.6539148   0.6492855
## Recall1 Calibration 0.6509000   0.6566811
## Recall2        Test 0.6606053   0.6481361
```

The logistic Regression model has AUC above 0.70 for Train, Calibration, and Test which is better NULL model. This has improved on previous single variable models. Recall has the value above 0.6 means that only 60% class correctly recognized by the model. This model can be used to help identify customer when they apply for a loan of their potential outcomes whether they might default or no.

```
#Visualization of AUC of Train, Calibration and Test Set.
roc.glm.train <- roc(dTrain$default,dTrain$pred)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

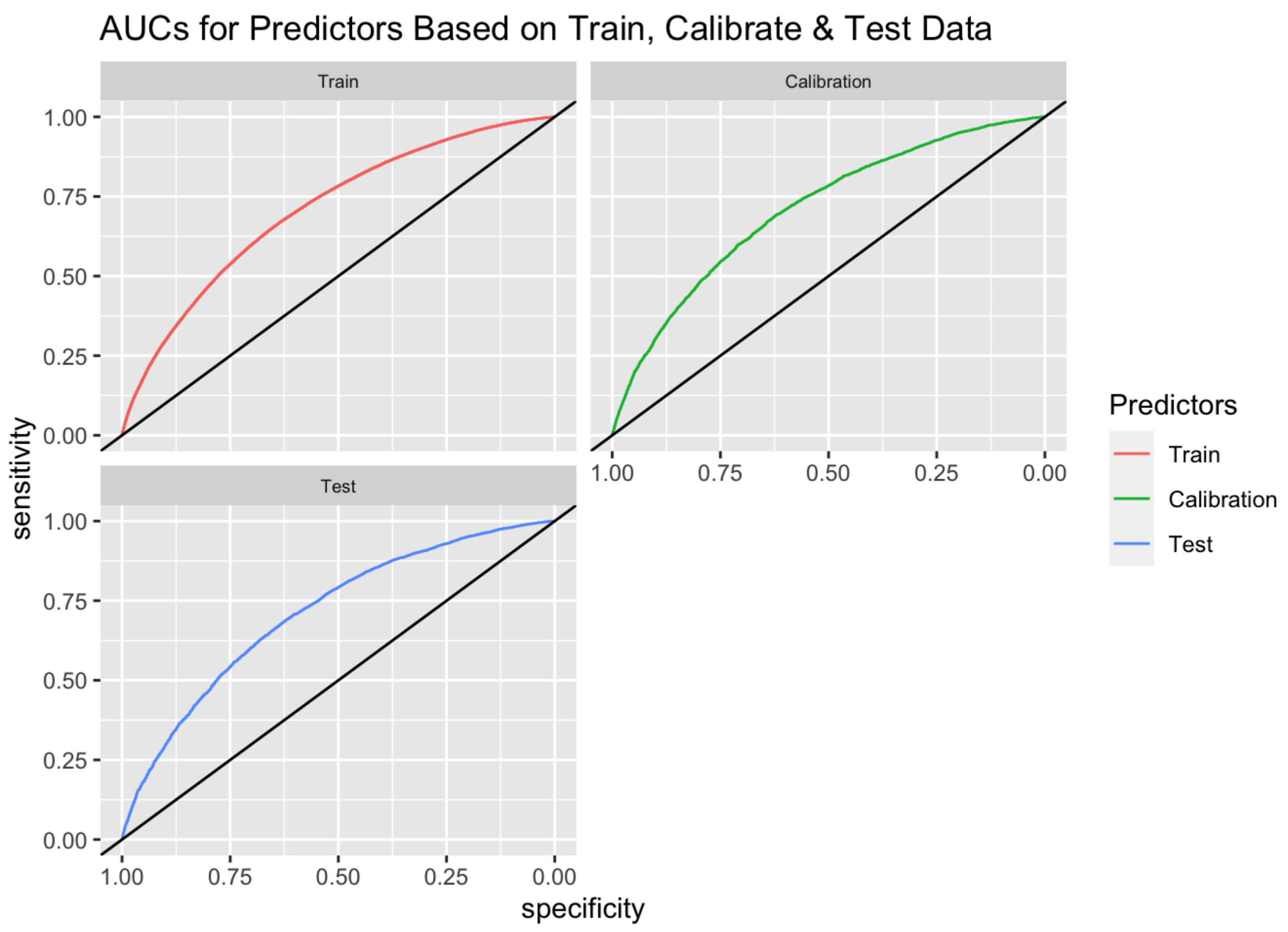
```
roc.glm.Cal <- roc(dCal$default,dCal$pred)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
roc.glm.Test <- roc(dTest$default,dTest$pred)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
roc.glm <- list(roc.glm.train,roc.glm.Cal,roc.glm.Test)

names(roc.glm)=c("Train","Calibration","Test")

ggroc(roc.glm)+
  geom_abline(intercept = 1, slope = 1,colour="black")+
  labs(title="AUCs for Predictors Based on Train, Calibrate & Test Data",colour="Pred
ictors")+
  theme(strip.text.x = element_text(size = 7))+facet_wrap(.~name,nrow=2,ncol=2)
```



As state before, AUC for Train, Calibration and Test for logistic model relatively the same at 0.7.

```
#Logistic with Selected Variable
(selVars <- c(selCatVars, selNumVars))
```

```
## [1] "pred_term"                 "pred_grade"
## [3] "pred_home_ownership"       "pred_verification_status"
## [5] "pred_annual_inc"           "pred_int_rate"
## [7] "pred_dti"                  "pred_revol_util"
## [9] "pred_mort_acc"
```

```
model_logr_reduce_var <- glm(formula=default ~ term + grade + home_ownership + verifi
cation_status + annual_inc + int_rate + dti + revol_util + mort_acc, data=dTrain, fam
ily=binomial(link="logit"))
dTrain$pred_reduce_var <- predict(model_logr_reduce_var, newdata=dTrain, type="respon
se")
dCal$pred_reduce_var <- predict(model_logr_reduce_var, newdata=dCal, type="response")
dTest$pred_reduce_var <- predict(model_logr_reduce_var, newdata=dTest, type="respons
e")
```

```
glm.aucTrain.reduce <- calcAUC(dTrain$pred_reduce_var,dTrain[,outcome])
glm.aucCal.reduce<- calcAUC(dCal$pred_reduce_var,dCal[,outcome])
glm.aucTest.reduce<- calcAUC(dTest$pred_reduce_var,dTest[,outcome])

print(sprintf("glm.trainAUC: %4.3f , glm.calAUC: %4.3f , glm.testAUC: %4.3f",
                glm.aucTrain.reduce, glm.aucCal.reduce, glm.aucTest.reduce))
```

```
## [1] "glm.trainAUC: 0.703 , glm.calAUC: 0.706 , glm.testAUC: 0.708"
```

```
roc.glm.train.reduce <- roc(dTrain$default,dTrain$pred_reduce_var)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

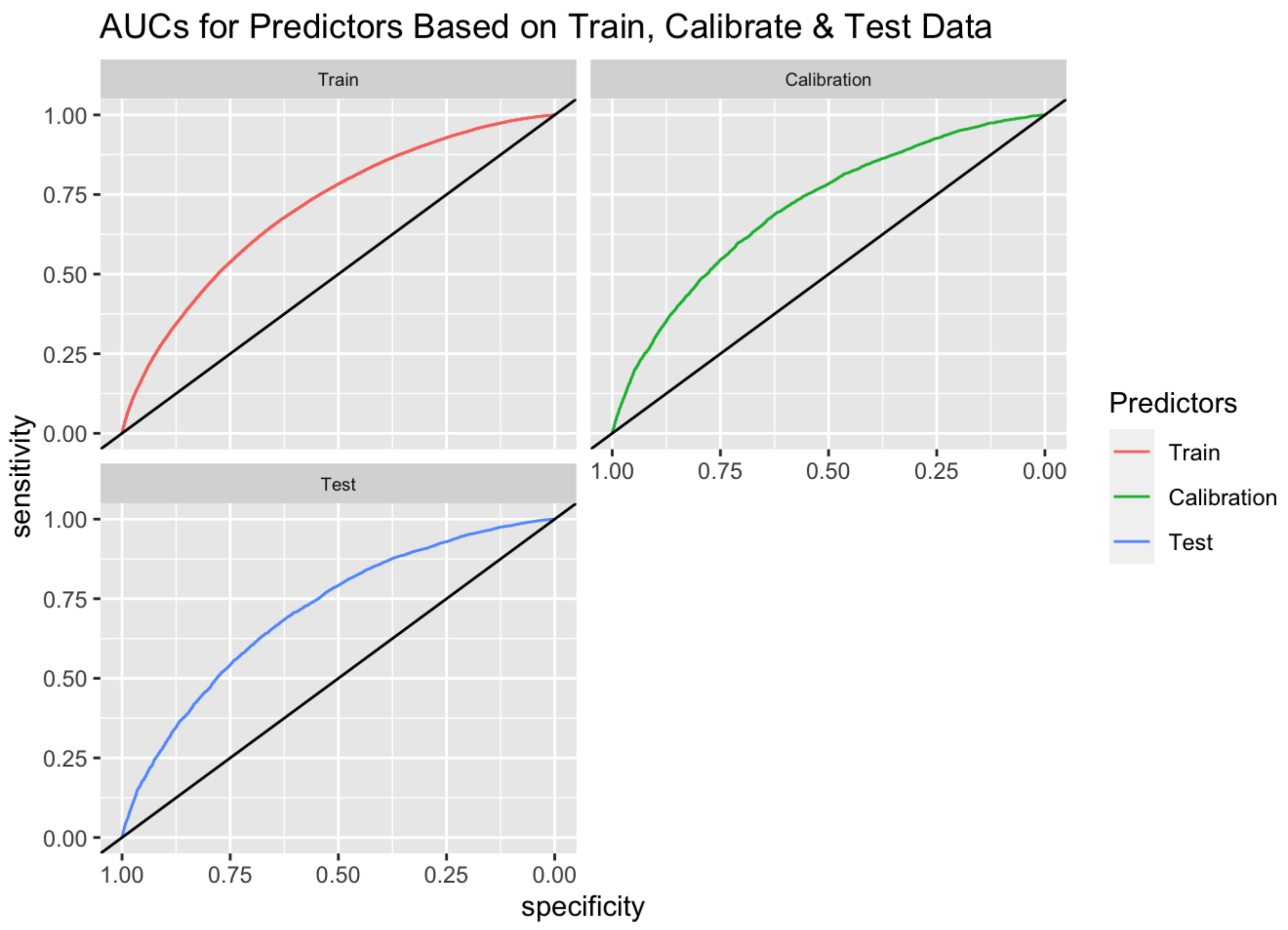
```
roc.glm.Cal.reduce <- roc(dCal$default,dCal$pred_reduce_var)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
roc.glm.Test.reduce <- roc(dTest$default,dTest$pred_reduce_var)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
roc.glm.reduce <- list(roc.glm.train.reduce,roc.glm.Cal.reduce,roc.glm.Test.reduce)

names(roc.glm.reduce)=c("Train","Calibration","Test")

ggroc(roc.glm)+
  geom_abline(intercept = 1, slope = 1,colour="black")+
  labs(title="AUCs for Predictors Based on Train, Calibrate & Test Data",colour="Pred
ictors")+
  theme(strip.text.x = element_text(size = 7))+facet_wrap(.~name,nrow=2,ncol=2)
```

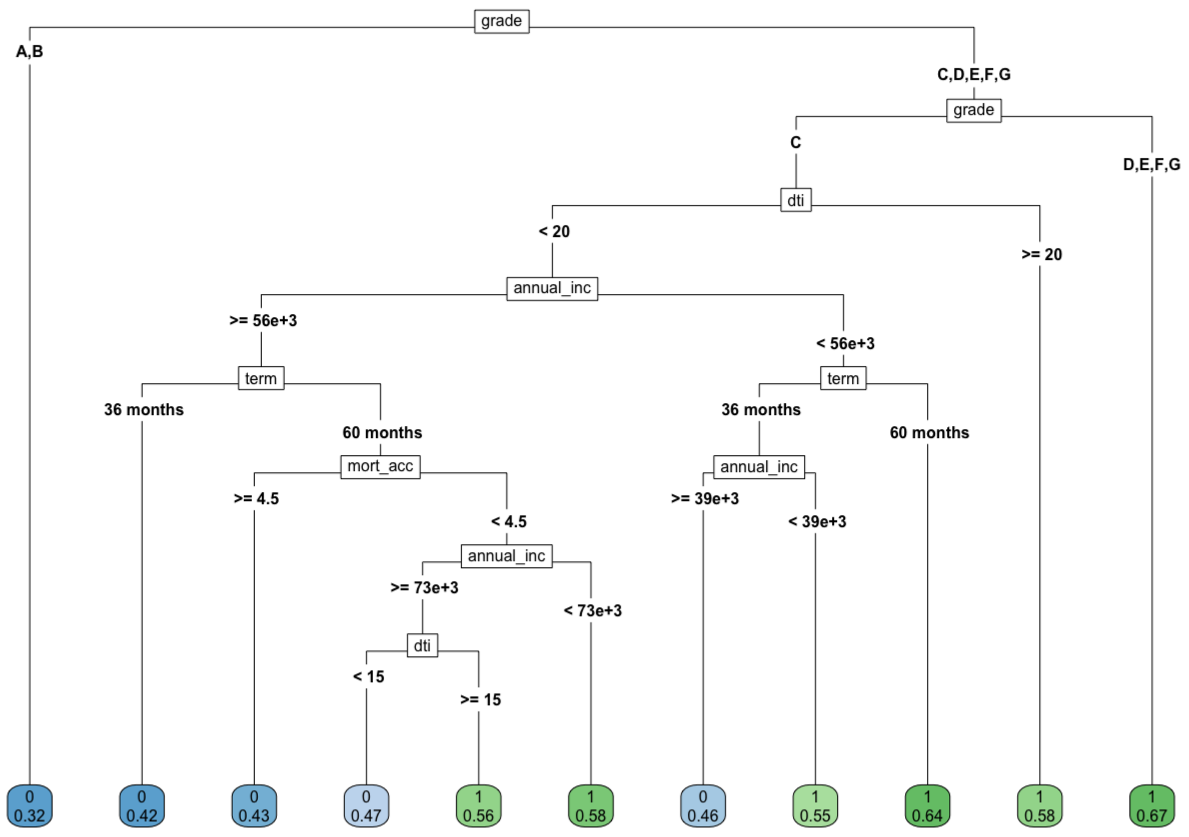


The logistic Regression model with selected variable has AUC of 0.7 it means that having less variables on the model doesn't change the performance of the model. Therefore, we will stick with the previous model with all of the variables.

4. Decision Tree Model

Decision Tree Model with be deployed to compare our existing model.

```
# decision tree model
treemodel <- rpart(fV, dTrain, method = "class",control=rpart.control(minsplit=10, mi
nbucket = 3, cp=0.0006))
rpart.plot(treemodel, type = 5, extra = 6)
```



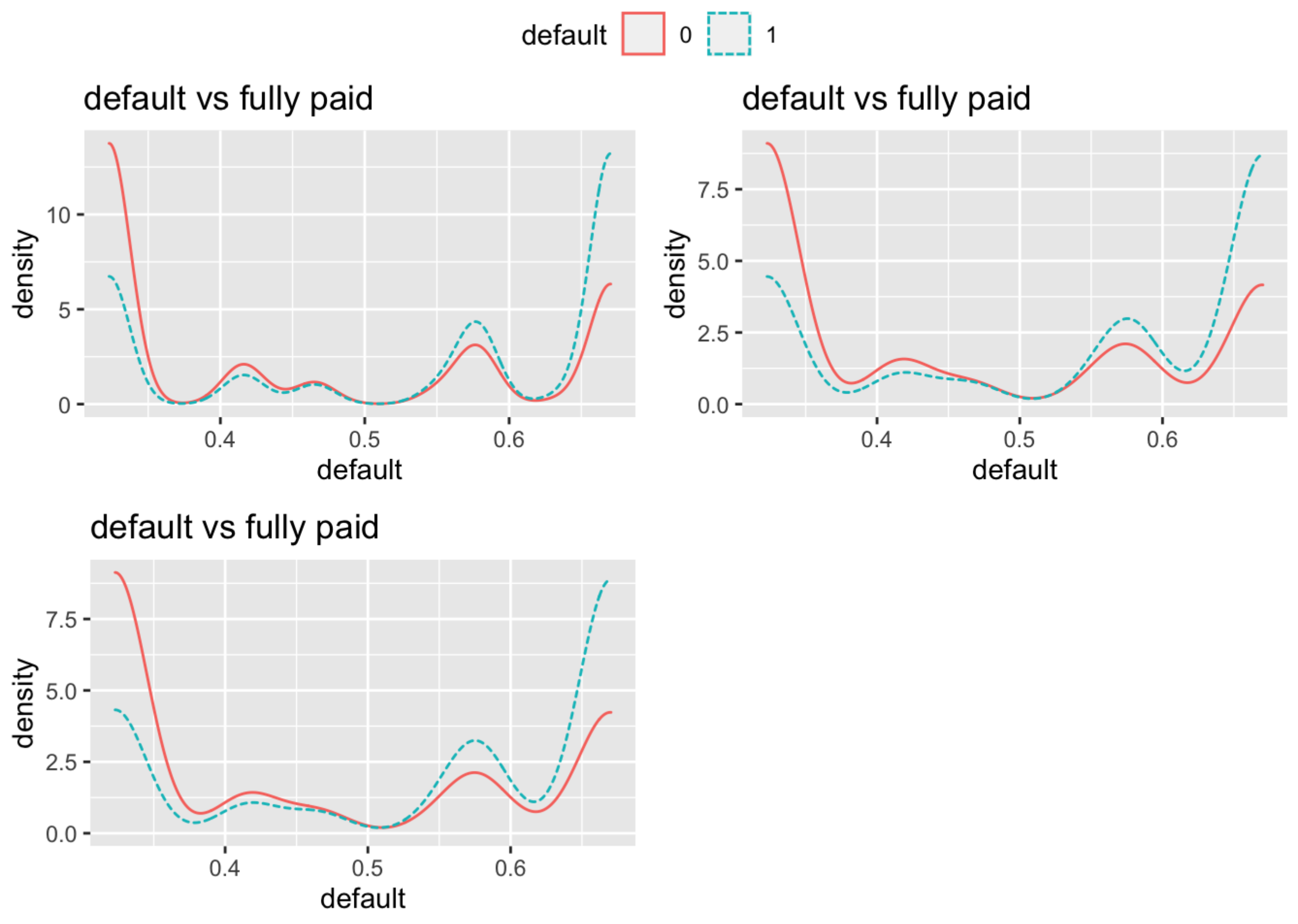
Based on the decision tree plot, variables that influence the outcome of default are : grade, dti, annual income, term and mort_account. Grade A and B are most likely to pay back the loan, whereas grade C,D,E,F,G might risk of defaulting the loan.

```
dTrain$pred_dt <- predict(treemodel, newdata=dTrain, type="prob")[,2]
dCal$pred_dt <- predict(treemodel, newdata=dCal, type="prob")[,2]
dTest$pred_dt <- predict(treemodel, newdata=dTest, type="prob")[,2]
```

```
dt.aucTrain <- calcAUC(dTrain$pred_dt,dTrain[,outcome])
dt.aucCal<- calcAUC(dCal$pred_dt,dCal[,outcome])
dt.aucTest<- calcAUC(dTest$pred_dt,dTest[,outcome])

  print(sprintf("dt.trainAUC: %4.3f , dt.calAUC: %4.3f , dt.testAUC: %4.3f",
              dt.aucTrain, dt.aucCal, dt.aucTest))
```

```
## [1] "dt.trainAUC: 0.667 , dt.calAUC: 0.668 , dt.testAUC: 0.671"
```

```
#Density plot to determine the likelihood treshold
train_dt_des<- ggplot(dTrain,aes(x=pred_dt,color=default,linetype=default))+geom_dens
ity()+
  labs(title="default vs fully paid",x="default",
       linetype="default",colour="default")

cal_dt_des<- ggplot(dCal,aes(x=pred_dt,color=default,linetype=default))+geom_density
()+
  labs(title="default vs fully paid",x="default",
       linetype="default",colour="default")

test_dt_des<- ggplot(dTest,aes(x=pred_dt,color=default,linetype=default))+geom_densit
y()+
  labs(title="default vs fully paid",x="default",
       linetype="default",colour="default")

ggarrange(train_dt_des,cal_dt_des,test_dt_des,
          font.label=list(size=5),
          vjust=0.5,
          ncol=2,nrow=2,
          common.legend = TRUE)
```



Based on the graph generated above, the prediction values showing the right class is relatively the same. The red line (non-default) has high density on lower probability for train, calibration and test set and the blue line (default) has high density on higher probability for train, calibration and test set.

```
#Function to measure the performance of the model
loglikelihood <- function(y, py) { pysmooth <- ifelse(py == 0, 1e-12,
ifelse(py == 1, 1 - 1e-12, py))
sum(y * log(pysmooth) + (1 - y) * log(1 - pysmooth)) }

accuracyMeasures <- function(pred, truth, name = "model") {
dev.norm <- -2 * loglikelihood(as.numeric(truth), pred) / length(pred)
ctable <- table(truth = truth, pred = (pred > 0.5))
accuracy <- sum(diag(ctable)) / sum(ctable)
precision <- ctable[2, 2] / sum(ctable[, 2])
recall <- ctable[2, 2] / sum(ctable[2, ])
f1 <- 2 * precision * recall / (precision + recall)
data.frame(model = name, accuracy = accuracy, f1 = f1, dev.norm)
}

trainperf_tree <- accuracyMeasures(dTrain$pred_dt, dTrain$default == "1", name = "tre
e, training")
calperf_tree <- accuracyMeasures(dCal$pred_dt, dCal$default == "1", name = "tree, Ca
l")
testperf_tree <- accuracyMeasures(dTest$pred_dt, dTest$default == "1", name = "tree,
test")

library(pander)
panderOptions("plain.ascii", TRUE)
panderOptions("keep.trailing.zeros", TRUE)
panderOptions("table.style", "simple")
perf_justify <- "lrrr"

perftable <- rbind(trainperf_tree, calperf_tree, testperf_tree)

pandoc.table(perftable, justify = perf_justify)
```

```
##
##
## model              accuracy       f1   dev.norm
## ---------------- ---------- -------- ----------
## tree, training       0.6459   0.6529      1.291
## tree, Cal            0.6461   0.6513      1.290
## tree, test           0.6516   0.6596      1.284
```

Decision Tree model has accuracy and f1 around 0.6, it means that the model predict 60% of correct class of default and harmonic mean of Precision and Recall which is provides a more accurate picture of instances that were erroneously categorized than the Accuracy Metric (F1).

```
#Visualizing AUC
roc.dt.train <- roc(dTrain$default,dTrain$pred_dt)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

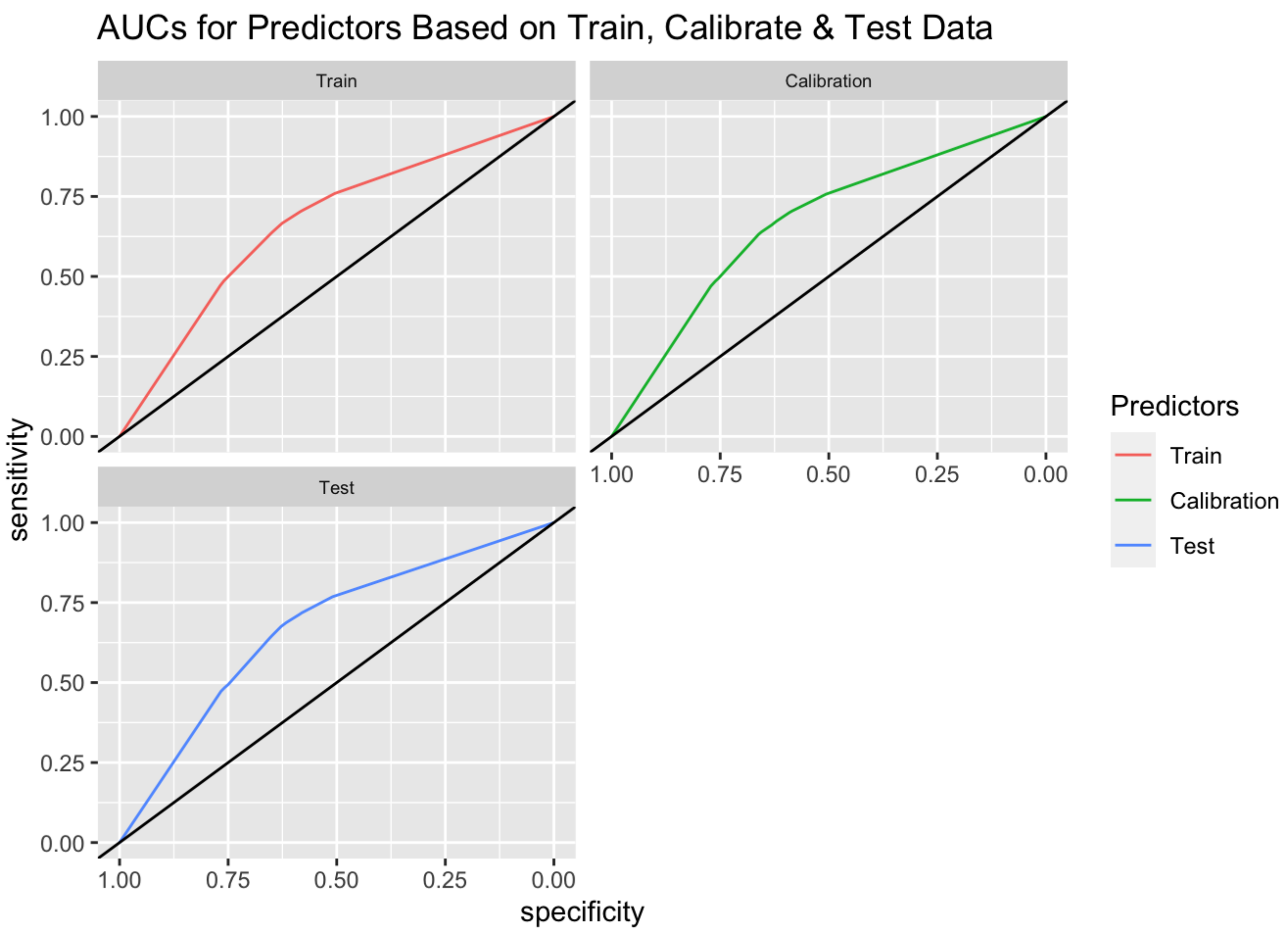
```
roc.dt.Cal <- roc(dCal$default,dCal$pred_dt)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
roc.dt.Test<- roc(dTest$default,dTest$pred_dt)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
roc.dt <- list(roc.dt.train, roc.dt.Cal, roc.dt.Test)

names(roc.dt)=c("Train","Calibration","Test")

ggroc(roc.dt)+
  geom_abline(intercept = 1, slope = 1,colour="black")+
  labs(title="AUCs for Predictors Based on Train, Calibrate & Test Data",colour="Pred
ictors")+
  theme(strip.text.x = element_text(size = 7))+facet_wrap(.~name,nrow=2,ncol=2)
```



Findings

Based on all the models that we performed, Logistic Regression showing the best results. The logistic Regression model has AUC above 0.70 for Train, Calibration, and Test which is better all NULL, Single, and Decision Tree model. Recall has the value above 0.6 means that 60% class correctly recognized by the model. This model can be used to help identify customer when they apply for a loan of their potential outcomes whether they might default or no.

Clustering

This section will include performing a cluster analysis by “groups” based on their proximity to one another. We utilise hierarchical clustering since we’re working with a distance matrix.We converted our dataset categorical variables into its projected probability figures, which implies the numbers are already scaled, because we have primarily categorical variables. The probabilities from our clustering study will be used to calculate the distance matrix.

```
rm(list = ls())
```

```
df_clustering<-read.csv(file ="lending_club_df_clustering.csv", header = TRUE, string
sAsFactors = TRUE)
df_clustering<- subset(df_clustering, select =c("term", "int_rate", "installment", "g
rade", "emp_length_merge", "home_ownership", "purpose_merge",  "verification_status",
"dti", "open_acc", "pub_rec", "revol_bal", "revol_util", "total_acc", "initial_list_s
tatus", "application_type", "pub_rec_bankruptcies"))
boxplot.stats(df_clustering$int_rate)$stats
```

```
## [1]  5.32 11.44 14.16 17.57 26.57
```

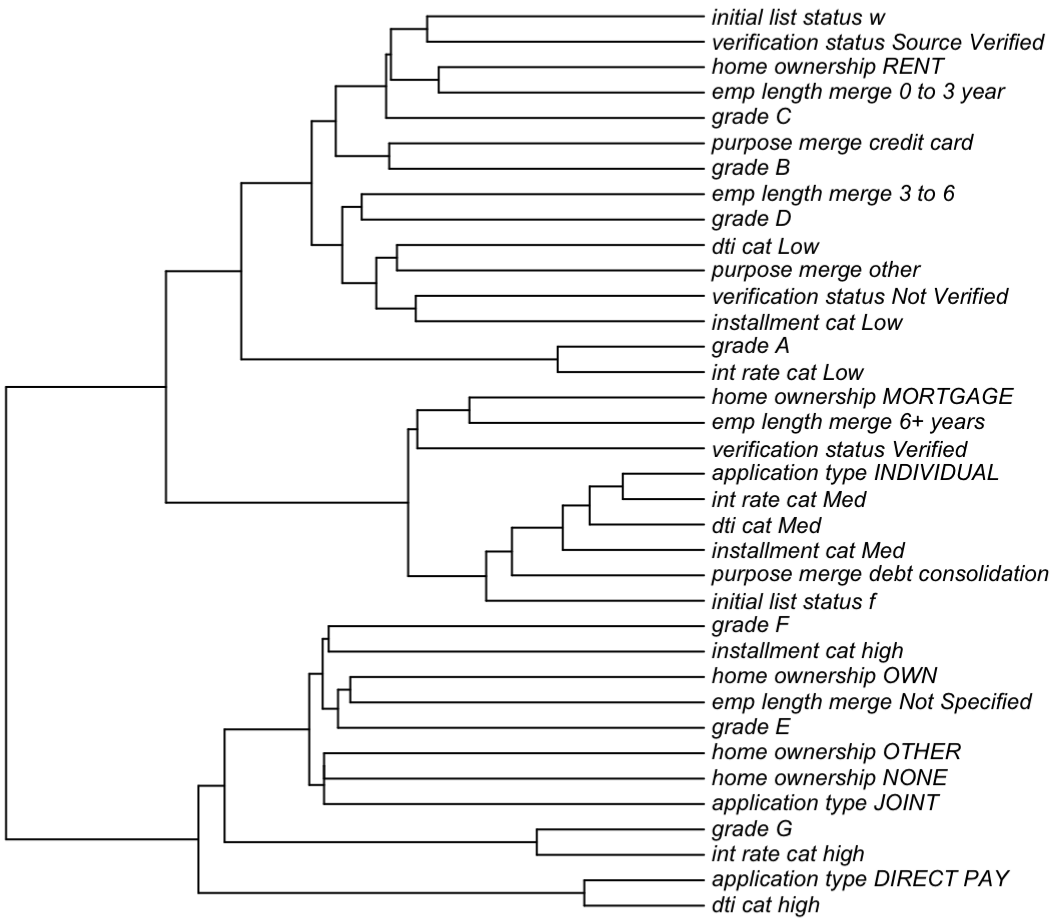
```
df_clustering$int_rate_cat <- cut(df_clustering$int_rate, breaks = c(-Inf, 10.49, 24.
99,Inf), labels = c("Low", "Med", "high"))
boxplot.stats(df_clustering$installment)$stats
```

```
## [1]   19.870  260.410  384.675  574.835 1046.000
```

```
df_clustering$installment_cat <- cut(df_clustering$installment, breaks = c(-Inf, 250.
33, 1043.97,Inf), labels = c("Low", "Med", "high"))
boxplot.stats(df_clustering$dti)$stats
```

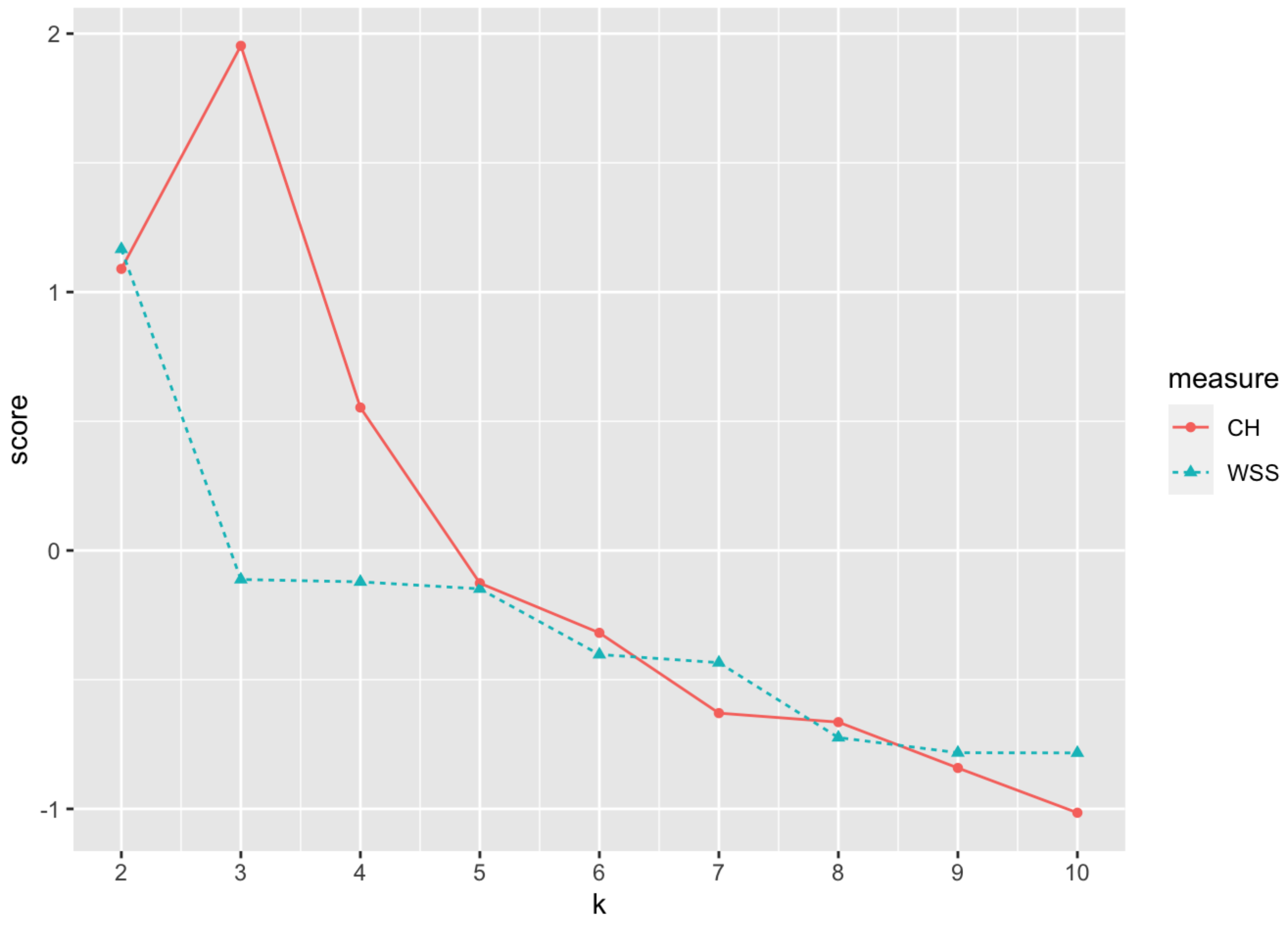
```
## [1]  0.00 12.02 17.80 24.02 41.88
```

```
df_clustering$dti_cat <- cut(df_clustering$dti, breaks = c(-Inf, 11.30, 40.52,Inf), l
abels = c("Low", "Med", "high"))

df_clustering_final <- subset(df_clustering, select =c("term", "int_rate_cat", "insta
llment_cat", "grade", "emp_length_merge", "home_ownership", "purpose_merge",  "verifi
cation_status", "dti_cat",  "initial_list_status", "application_type"))
df_clustering_final <- df_clustering_final[,-1]
ohe.data <- one_hot(as.data.table(df_clustering_final),naCols=TRUE)
#tr
ohe.data.t <- t(ohe.data)
dist.matrix <- dist(ohe.data.t, method="binary")
lending_hclus <- hclust(dist.matrix, method="ward.D2")
par(mar = c(0, 0, 0, 0))
plot(as.phylo(lending_hclus), cex = 0.7, label.offset = 0.01)
```



The optimal number of clusters will next be determined using the WSS and CH index as indicators.

```
# Function to calculate squared distance
# between two vectors x and y
sqr_edist <- function(x, y) {
sum((x-y)^2)
}
## Function to calculate WSS of a cluster
wss.cluster <- function(clustermat) {

  c0 <- apply(clustermat, 2, FUN=mean)
  sum(apply(clustermat, 1,
  FUN=function(row){sqr_edist(row,c0)}))
}

wss.total <- function(dmatrix, labels) {
  wsstot <- 0
  k <- length(unique(labels))
  for(i in 1:k) {
    wsstot <- wsstot +
    wss.cluster(subset(dmatrix, labels==i))
  }
  wsstot
}

totss <- function(dmatrix) {
  grandmean <- apply(dmatrix, 2, FUN=mean)
  sum(apply(dmatrix, 1,
  FUN=function(row){
    sqr_edist(row, grandmean) }))
}

ch_criterion <- function(dmatrix, kmax, method="kmeans") {
  if(!(method %in% c("kmeans", "hclust"))){
    stop("method must be one of c('kmeans', 'hclust')")
  }

  npts <- dim(dmatrix)[1] # number of rows.
  totss <- totss(dmatrix)
  wss <- numeric(kmax)
  crit <- numeric(kmax)

  wss[1] <- (npts-1)*sum(apply(dmatrix, 2, var))

  for(k in 2:kmax) {
    if(method=="kmeans") {
      clustering<-kmeans(dmatrix, k, nstart=10, iter.max=100)
      wss[k] <- clustering$tot.withinss
    }
    else { # hclust
      d <- dist(dmatrix, method="binary")
      pfit <- hclust(d, method="ward.D2")
      labels <- cutree(pfit, k=k)
      wss[k] <- wss.total(dmatrix, labels)
     }
  }
  bss <- totss - wss
  crit.num <- bss/(0:(kmax-1))
  crit.denom <- wss/(npts - 1:kmax)
  list(crit = crit.num/crit.denom, wss = wss, totss = totss)
}

clustcrit <- ch_criterion(ohe.data.t, 10, method="hclust")

CH_index <- data.frame(k=2:10,measure="CH",score=scale(clustcrit$crit[2:10]))

wss_index <- data.frame(k=2:10,measure="WSS",score=scale(clustcrit$wss)[2:10])

wv_index <- data.frame(k=2:10)

clus_criteria <- rbind(CH_index,wss_index)

ggplot(clus_criteria, aes(x=k, y=score, color=measure)) +
geom_point(aes(shape=measure)) +
geom_line(aes(linetype=measure)) +
scale_x_continuous(breaks=2:10, labels=2:10)
```



The score of CH showing that k cluster = 3 has the highest score for WSS, the highest cluster was k = 2. We'll additionally look at 8 clusters since their scaled CH index is larger than their scaled WSS.

```
hclus3 <- as.factor(cutree(lending_hclus,k=3))

hclus4 <- as.factor(cutree(lending_hclus,k=4))

hclus6 <- as.factor(cutree(lending_hclus,k=6))

print_clusters <- function(clusters){
  for(i in 1:length(levels(clusters))){
    cat(paste0('Cluster ',i,':\n'))
    cat(paste(names(clusters[clusters == i]), collapse = "\t"))
    cat('\n\n')
  }
}

print_clusters(hclus3)
```

```
## Cluster 1:
## int_rate_cat_Low installment_cat_Low grade_A grade_B grade_C grade_D emp_length_me
rge_0 to 3 year    emp_length_merge_3 to 6 home_ownership_RENT purpose_merge_credit_c
ard   purpose_merge_other verification_status_Not Verified    verification_status_Sou
rce Verified dti_cat_Low initial_list_status_w
##
## Cluster 2:
## int_rate_cat_Med installment_cat_Med emp_length_merge_6+ years   home_ownership_MO
RTGAGE purpose_merge_debt_consolidation    verification_status_Verified    dti_cat_Me
d initial_list_status_f   application_type_INDIVIDUAL
##
## Cluster 3:
## int_rate_cat_high    installment_cat_high    grade_E grade_F grade_G emp_length_me
rge_Not Specified  home_ownership_NONE home_ownership_OTHER    home_ownership_OWN  dt
i_cat_high    application_type_DIRECT_PAY application_type_JOINT
```

```
table(hclus3)
```

```
## hclus3
##  1  2  3
## 15  9 12
```

Based on the cluster, 1st cluster seems to identify applicants with the most likely outcome of non-default. This is because the character of the variables in the cluster are most likely the character of applicants who has good credit score which implied from the Grade of the loan they received and has interest rate of low which is implied that they are safe applicants (the chance of default is low). On the other hand, 3rd cluster identify applicants with the chance of defaulting. This is can be seen from the grade or E, F and G which is showing that they don't have good credit score and they don't specify the length of employment means that they might not has a paying job at the time of applying and also it's worth noting that type application is Joint means that Joint application with lower grade increase chance of defaulting.

```
print_clusters(hclus4)
```

```
## Cluster 1:
## int_rate_cat_Low installment_cat_Low grade_A grade_B grade_C grade_D emp_length_me
rge_0 to 3 year    emp_length_merge_3 to 6 home_ownership_RENT purpose_merge_credit_c
ard   purpose_merge_other verification_status_Not Verified    verification_status_Sou
rce Verified dti_cat_Low initial_list_status_w
##
## Cluster 2:
## int_rate_cat_Med installment_cat_Med emp_length_merge_6+ years   home_ownership_MO
RTGAGE purpose_merge_debt_consolidation    verification_status_Verified    dti_cat_Me
d initial_list_status_f   application_type_INDIVIDUAL
##
## Cluster 3:
## int_rate_cat_high    installment_cat_high    grade_E grade_F grade_G emp_length_me
rge_Not Specified  home_ownership_NONE home_ownership_OTHER    home_ownership_OWN  ap
plication_type_JOINT
##
## Cluster 4:
## dti_cat_high application_type_DIRECT_PAY
```

```
table(hclus4)
```

```
## hclus4
##  1  2  3  4
## 15  9 10  2
```

```
print_clusters(hclus6)
```

```
## Cluster 1:
## int_rate_cat_Low grade_A
##
## Cluster 2:
## int_rate_cat_Med installment_cat_Med emp_length_merge_6+ years   home_ownership_MO
RTGAGE purpose_merge_debt_consolidation    verification_status_Verified    dti_cat_Me
d initial_list_status_f   application_type_INDIVIDUAL
##
## Cluster 3:
## int_rate_cat_high    grade_G
##
## Cluster 4:
## installment_cat_Low  grade_B grade_C grade_D emp_length_merge_0 to 3 year    emp_l
ength_merge_3 to 6 home_ownership_RENT purpose_merge_credit_card   purpose_merge_othe
r verification_status_Not Verified    verification_status_Source Verified dti_cat_Low
initial_list_status_w
##
## Cluster 5:
## installment_cat_high grade_E grade_F emp_length_merge_Not Specified  home_ownershi
p_NONE home_ownership_OTHER    home_ownership_OWN  application_type_JOINT
##
## Cluster 6:
## dti_cat_high application_type_DIRECT_PAY
```

```
table(hclus6)
```

```
## hclus6
##  1  2  3  4  5  6 
##  2  9  2 13  8  2
```

Conclusion

We utilised the Lending Club dataset to fit a predictive classifier that can predict whether an application would pay back the loan in full or default based on their profile. We can predict 70% of the correct outcomes using our approach. Although not perfect, this is a good start to forecasting, and perhaps in the future we will be able to enhance our model with a better algorithm and higher data quality.

Appendix

```
LoanStatNew     Description
```

0 loan_amnt - The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
1 term - The number of payments on the loan. Values are in months and can be either 36 or 60.
2 int_rate - Interest Rate on the loan
3 installment - The monthly payment owed by the borrower if the loan originates.
4 grade - LC assigned loan grade
5 sub_grade - LC assigned loan subgrade
6 emp_title - The job title supplied by the Borrower when applying for the loan.*
7 emp_length -Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
8 home_ownership - The home ownership status provided by the borrower during registration or obtained from the credit report. Our values are: RENT, OWN, MORTGAGE, OTHER
9 annual_inc - The self-reported annual income provided by the borrower during registration.
10 verification_status - Indicates if income was verified by LC, not verified, or if the income source was verified
11 issue_d - The month which the loan was funded
12 loan_status - Current status of the loan
13 purpose - A category provided by the borrower for the loan request.
14 title - The loan title provided by the borrower
15 zip_code - The first 3 numbers of the zip code provided by the borrower in the loan application.
16 addr_state -The state provided by the borrower in the loan application
17 dti - A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.
18 earliest_cr_line - The month the borrower's earliest reported credit line was opened
19 open_acc - The number of open credit lines in the borrower's credit file.
20 pub_rec - Number of derogatory public records
21 revol_bal - Total credit revolving balance
22 revol_util - Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
23 total_acc - The total number of credit lines currently in the borrower's credit file
24 initial_list_status - The initial listing status of the loan. Possible values are – W, F
25 application_type - Indicates whether the loan is an individual application or a joint application with two co-

borrowers
26 mort_acc - Number of mortgage accounts.
27 pub_rec_bankruptcies - Number of public record bankruptcies